Quasiparticle Tunneling and High Bias Breakdown in the Fractional Quantum Hall Effect

Thesis advisor

Marc Kastner

Author

Colin Ramsay Dillard

Quasiparticle Tunneling and High Bias Breakdown in the Fractional Quantum Hall Effect

Abstract

The integer and fractional quantum Hall effects arise in two-dimensional electron systems subject to low temperature and high perpendicular magnetic field. The phenomenology of these two effects is rich and provides interesting insight into quantum physics. We present two experimental studies of phenomena in the fractional quantum Hall regime.

The first examines the tunneling conductance of quasiparticles at filling factor 5/2. This state is of significant interest because it lies outside the traditional Jain hierarchy of fractional quantum Hall states and because it may be the first physical system found to exhibit non-abelian particle statistics. A quantum point contact is used to bring edge states on opposite sides of the system in proximity to each other, allowing quasiparticles to tunnel between the edge states. By annealing the gates forming the quantum point contact at different voltages we control the tunneling strength for fixed temperature and bias. We demonstrate a transition from strong to weak tunneling controlled in this manner. In the weak tunneling regime, the DC bias and temperature dependence of the tunneling conductance is fit to a theoretical form, resulting in values for the quasiparticle charge e^* and the interaction parameter g. The values of these parameters are used to help distinguish between proposed candidate states for the

5/2 wave function. Quantitative and qualitative results are most consistent with the abelian 331 state.

Our second main focus is the breakdown of the fractional quantum Hall states at filling factors 4/3 and 5/3. Breakdown of integer and fractional quantum Hall states is known to occur when the Hall and longitudinal resistances deviate from their ideal values at nonzero critical currents. Although multiple studies of breakdown in the integer quantum Hall regime have been reported, corresponding results for the fractional regime are scarce. We observe breakdown over a range of integer states that is consistent with previous results. However, breakdown in the fractional regime is found to exhibit markedly different behavior. In particular, the magnitude of the critical current decreases with increased sample width. This behavior is opposite that observed for integer filling factors and does not seem to be explicable based on current theories of breakdown.

Contents

Title Page i
Abstract iii
Table of Contents v
List of Figures vii
List of Tables ix
Citations to Published Work x
Acknowledgements xi

1 Introduction to the Quantum Hall Effect **1**
1.1 Overview 1
1.2 Classical Hall Effect 3
1.3 Integer Quantum Hall Effect 4
1.4 Fractional Quantum Hall Effect 12
1.5 Edge State Picture 24

2 Experimental Methods **34**
2.1 Experimental Details 34
2.2 Gate Annealing Technique 43
2.3 QPC Width and Electron Density 48

3 Quasiparticle Tunneling at $\nu = 5/2$ **56**
3.1 Controlling Tunneling Strength with Different Annealing Voltages . . 56
3.2 Tunneling Conductance in the Weak Tunneling Regime 67

4 Breakdown at Various Filling Factors **85**

A List of Symbols **108**

B Device Fabrication Procedures **114**
B.1 Fabrication Procedures 114
B.2 Electron Beam Lithography Procedure 121

C Code for Data Collection and Analysis **128**
C.1 General Approach to Data Collection and Storage 130
C.2 Initializations 133
C.3 Agilent Voltage Source 135
C.4 Yokogawa Voltage Source 141
C.5 Magnet Power Supply 145
C.6 Digital Multimeter 150
C.7 Data Collection 152
C.8 Data Handling 219
C.9 Data Searching 229
C.10 Physical Constants 242
C.11 Data Analysis 244
C.12 Fit Functions 248

Bibliography **251**

List of Figures

1.1 The integer quantum Hall effect. 11
1.2 Discovery of the fractional quantum Hall effect. 13
1.3 Various fractional quantum Hall states. 18
1.4 The Jain hierarchy. 21
1.5 Edge states in a simple Hall bar. 27
1.6 Edge states in a QPC. 30

2.1 Common GaAs/AlGaAs heterostructures. 36
2.2 Diagram of experimental Hall bar and SEM image of gate geometry. . 39
2.3 Depletion and pinch off of a QPC with gate voltage. 40
2.4 Diagram of experimental Hall bar with illustration of electrical measurements performed. 42
2.5 Effect of annealing gates at low and high magnetic fields. 45
2.6 Differential resistances and fits at low magnetic field. 51
2.7 QPC width and electron density extracted from fits to r_D with gates not annealed. 54
2.8 QPC width and electron density extracted from fits to r_D with gates annealed. 55

3.1 Magnetic field dependence of r_{XX}, r_{XY}, and r_D between the $\nu = 3$ and 2 plateaus. 58
3.2 Differential resistances and fits at low magnetic field and various gate voltages. 61
3.3 DC bias and temperature dependence of r_D for three different annealing voltages. 64
3.4 Temperature scaling of data from Figure 3.3. 66
3.5 DC bias and temperature dependence of r_D for two gate geometries. . 70
3.6 Differential resistance r_D as a function of DC bias current and gate voltage for Geometry B. 71
3.7 Scaling of peak height and full width at half-maximum with temperature. 74

3.8 Least-squares best fit of the theoretical form of weak tunneling conductance to r_D. 75
3.9 Matrix of fit residual divided by the experimental noise for fixed pairs of (e^*, g). 77
3.10 Matrix of reduced chi-squared of fits for fixed pairs of (e^*, g). 78
3.11 Fits of the theoretical form of quasiparticle weak tunneling to r_D for Geometry A. 80
3.12 Fits of the theoretical form of quasiparticle weak tunneling to r_D for Geometry B. 81

4.1 Example of breakdown in the QPC at nonzero critical currents. . . . 93
4.2 Examples of two different types of hysteresis. 95
4.3 Critical current as a function of magnetic field for various filling factors. 97
4.4 Power law fits of critical current dependence on magnetic field. 98
4.5 Example of breakdown at an integer quantum Hall plateau. 100
4.6 Critical current as function of gate voltage at various filling factors. . 102
4.7 Critical current as function of QPC width at various filling factors. . 105
4.8 Temperature dependence of critical current versus magnetic field and gate voltage. 106

List of Tables

3.1 Summary of QPC widths extracted by fits to data in Figure 3.2. . . . 62

4.1 Exponents extracted by power law fits using Equation (4.2) at integer filling factors. 99

Citations to Published Work

The the results of Section 3.2 have been published as:

> "Measurements of quasiparticle tunneling in the $\nu = 5/2$ fractional quantum Hall state", X. Lin, C. Dillard, M. A. Kastner, L. N. Pfeiffer, and K. W. West, Phys. Rev. B. **85**, 165321 (2012);

and those of Chapter 4 have been submitted for publication as:

> "Breakdown of the integer and fractional quantum Hall effects in a quantum point contact", C. Dillard, X. Lin, M. A. Kastner, L. N. Pfeiffer, and K. W. West, submitted to Phys. Rev. B.
> `http://arXiv.org/abs/1204.5712`.

The author contributed to the following work, not discussed in this thesis:

> "The effect of surface conductance on lateral gated quantum devices in Si/SiGe heterostructures", Xi Lin, Jingshi Hu, Andrew P. Lai, Zhenning Zhang, Kenneth MacLean, Colin Dillard, Ya-Hong Xie, and Marc A. Kastner, J. Appl. Phys. **110**, 023712 (2011).

Acknowledgements

I completed my graduate work in Marc Kastner's lab at MIT. I am grateful to Marc for accepting me into his group and guiding me through the process of performing experimental physics research. I appreciate the freedom and flexibility he allows his graduate students to explore the physics we find interesting. And I hope some of his strong intuition and understanding of physics has rubbed off on me. I would also like to thank the other members of my committee: Isaac Silvera (my academic advisor at Harvard), Amir Yacoby, and Bertrand Halperin.

The data presented in this thesis was collected by myself and Xi Lin, who spent a few years as a postdoctoral associate in our group. Xi was really the one who kept the dilution refrigerator running and he also contributed significantly to chip fabrication and data analysis. I greatly appreciate the time and effort he spent keeping an eye on the fridge at all hours day and night.

I worked with Emily Davis, who joined the lab as part of the UROP program at MIT, for two years as she helped improve and expand our fabrication recipes. The implementation of the etch step to create trenches under the small gates was primarily a result of her work. Although the process was far from easy at times (cleanroom equipment has a nasty habit of breaking when one needs it the most) she persevered

and did the majority of the fabrication of a chip with which we hope to perform the next set of $\nu = 5/2$ experiments.

In my early years in the Kastner lab I worked with Iuliana Radu (a graduate student) and Jeff Miller (a postdoc in the Marcus group at Harvard, with whom we collaborated), helping them to measure tunneling conductance in a simple QPC at $\nu = 5/2$. The experimental technique for that measurement as well as the basics of the gate annealing technique were developed/discovered by Iuliana and Jeff. Much of what I learned about performing experimental condensed matter physics came from working with the two of them. Over the years, I have also worked with or in close proximity to a number of other people in the Kastner lab: Ghislain Granger, Dominik Zumbühl, Sami Amasha, Kenny MacLean, Ian Gelfand, Jingshi Hu, Tamar Mentzel, Nirat Ray, and Andrew Lai. I also worked briefly with Christian Barthel in the Marcus group. I'm grateful to all these people for the help and instruction each gave me. Kenny deserves special thanks for helping Xi and I calibrate the electron temperature using thermally-broadened Coulomb blockade peaks.

Loren Pfeiffer and Ken West grow high quality GaAs heterostructures using molecular beam epitaxy and have been very generous to give a number of pieces from various wafers to our group. Xi and I performed a number of experiments, not reported on in this thesis, using chips fabricated from those heterostructures. For the data in this thesis, the heterostructure was also provided by Loren and Ken and the chip was fabricated by Jeff Miller while he was part of the Marcus group. I greatly appreciate Charlie's willingness to loan us the chip for Xi and me to measure again. I would also like to thank him, as well as Doug McClure and Angela Kou (graduate students in

the Marcus group) for letting us use their dilution refrigerator for a few weeks during a time that ours was not functional. Doug has also been generous with sharing chip fabrication recipes and tips; the etch procedure implemented by Emily was adapted from a recipe from Doug.

I have had a number of conversations about various experimental results with Charlie Marcus, Claudio Chamon, Dmitri Feldman, and Xiao-Gang Wen and would like to thank all of them for their time and effort to help me understand our data. In addition, Paul Fendley and Chetan Nayak contributed to our understanding of tunneling at $\nu = 5/2$. Bas Overbosch, a graduate student of Xiao-Gang, was instrumental in translating the results of the tunneling conductance paper [1] into the function cited in this thesis and used to fit our results.

On a personal note, I'd like to thank my parents, Annette and Robert Dillard, and also MinJi Kim for proofreading this thesis and for encouraging me to pursue my dreams.

The work presented in this thesis was funded by National Science Foundation under Grant No. DMR-1104394.

Chapter 1

Introduction to the Quantum Hall Effect

1.1 Overview

The integer and fractional quantum Hall effects (QHEs) play host to a great deal of interesting physics. The key features observed experimentally—a quantized transverse (Hall) resistance and a vanishing longitudinal resistance—are fundamentally the result of quantum mechanics. The fact that these features are observed in macroscopic samples with high amounts of disorder is perhaps rather nonintuitive and has encouraged the current trend of analyzing condensed matter systems from a topological perspective. This perspective leads to the picture of one-dimensional edge states, which in the fractional QHE are described as chiral Luttinger liquids [2]. The quasiparticles which form the fractional QHE exhibit unique particle statistics, allowing arbitrary phases factors to accumulate in the wave function upon particle

exchange [3], in contrast to the traditional factor of ± 1 for bosons and fermions. Even more strangely, the fractional quantum Hall state at filling factor 5/2 may exhibit non-abelian statistics [4], in which the wave function changes completely when two particles are exchanged. Other QHE phenomena, such as the reentrant integer QHE and proposed stripe and bubble phases, are still largely unexplored. We discuss the origin of the integer and fractional QHEs and some of their phenomenology in the remaining sections of this chapter.

Many interesting experiments related to the QHE are performed in regimes in which, in some sense, the ideal behavior described above is deliberately destroyed. One common technique is to induce tunneling of quasiparticles across a quantum point contact (QPC). By allowing backscattering, quantization of the Hall resistance is lost, at least in the QPC. However, this technique allows one to probe the physics of the quasiparticles and the edge states which carry them. A number of experiments have been performed utilizing this technique. We present measurements of the quasiparticle tunneling conductance across a QPC in Chapter 3. Although the experiment is not able to directly probe the presence of abelian versus non-abelian statistics, our results do provide information to help distinguish between possible candidate states at filling factor 5/2. Shot noise produced by quasiparticle tunneling across a QPC has also been measured [5]. Other experiments have studied interferometer devices, formed by placing two QPCs in series [6, 7]. The observed interference pattern is determined by a combination of the Aharonov-Bohm phase and the statistical phase of the quasiparticles [8]. In particular, states with non-abelian statistics should exhibit a distinct interference pattern [9]. Initial results are roughly consistent with

the expected non-abelian signature [10, 11]. As these experimental techniques are refined, they have been adapted to provide new methods of probing QHE physics, for example, detection of neutral edge states [12, 13].

Another regime in which ideal QHE behavior is destroyed is that of breakdown due to high current bias. The mechanism causing this breakdown and the behavior at currents close to breakdown are of interest in order to guarantee that the international standard of resistance defined by the integer QHE is as accurate as possible. Understanding breakdown of the QHE may also increase our understanding of the effect itself and, in particular, the limits of the phase space in which we would expect ideal behavior. A number of experiments have studied breakdown of the integer QHE [14]; however there has been only minimal work done in the fractional regime. In Chapter 4 we present a set of measurements spanning states in both the integer and fractional regimes. Interestingly, we find previously unobserved signatures in the fractional regime which are opposite those observed in the integer regime.

1.2 Classical Hall Effect

The classical Hall effect was discovered by Edwin Hall in 1989. It arises in an electrical conductor in which a uniform magnetic field $B_z = B$ is applied perpendicular to the direction of current flow. The Lorentz force acting on the charge carriers is

$$\mathbf{F} = q(\mathbf{E} + \mathbf{v} \times \mathbf{B}).$$

Let us take the current to be limited (by the boundary of the sample) to the x direction and the magnetic field to be in the z direction. Because there is no current

in the y direction, that component of the force is zero and we have $E_y = vB$. For a sample with a rectangular cross-section of area $A = L_y L_z$ the total current $I = qnAv$, where n is the density of charge carriers and v is their drift velocity. This sample experiences a nonzero Hall voltage $V_H = E_y L_y$ perpendicular to the current,

$$V_H = \frac{IB}{qnL_z}.$$

For constant current, the Hall voltage is proportional to the perpendicular magnetic field, with a slope that depends on the carrier charge and density and the depth L_z of the sample.

If we consider a hypothetical two-dimensional sample (as we will for the QHE), the dimensions of the sample cancel out, giving $V_H = IB/qn$. In analogy to the longitudinal resistance, we define the Hall resistance $R_H = V_H/I$. In this thesis we only study samples with electrons as the charge carriers; hence we will take $|q| = e$ to be the electric charge and redefine the polarity of R_H so it is positive for positive B:

$$R_H \equiv \frac{V_H}{I} = \frac{B}{en}. \tag{1.1}$$

1.3 Integer Quantum Hall Effect

We now consider a two-dimensional system of electrons at low temperatures and under high perpendicular magnetic field, such that quantum effects must be considered. Practically, electrons may be confined to two dimensions by a potential well formed by the band offset in a semiconductor heterostructure. For strong confinement and low temperatures, the electrons are limited to the ground state in this well,

hence localizing them to two dimensions. The implementation of this technique in GaAs/AlGaAs heterostructures is discussed in Section 2.1.

Ignoring spin, the Hamiltonian of the system is

$$H = \frac{1}{2m^*}\left(p_y^2 + (p_x - eBy)^2 + p_z^2\right) + V(z). \tag{1.2}$$

Here we have used the Landau gauge ($\mathbf{A} = -By\hat{\mathbf{x}}$) and the effective electron mass $m^* = 0.067m_e$ (for GaAs). The z component of the wave function separates out and, for low temperatures, is limited to the ground state of the potential $V(z)$. The time-independent Schrödinger equation can be solved by wave functions of the form $\Psi(x, y) = f(y)e^{ik_x x}$. These are plane waves in the x direction with some, as yet, unknown distribution in the y direction. Using this wave function replaces the p_x operator by $\hbar k_x$, giving the Hamiltonian the form of a harmonic oscillator centered at $y_0 = \hbar k_x/eB$ with frequency equal to the classical cyclotron frequency $\omega_c = eB/m^*$:

$$H_{xy} = \frac{p_y^2}{2m^*} + \frac{m^*\omega_c^2}{2}(y - y_0)^2. \tag{1.3}$$

Hence, the eigenstates of the system are

$$\Psi_{j,k_x}(x, y) = e^{ik_x x}\phi_j(y - y_0), \tag{1.4}$$

with ϕ_j the normalized eigenstates of the harmonic oscillator. For a given quantum number j, there is an infinite degeneracy of plane waves in the x direction, each localized in the y direction with an offset proportional to their x momentum. The energy of the states is given by $E_j = \hbar\omega_c(j + 1/2)$. Changing gauge will change the form of the eigenstates (for example, in the symmetric gauge they have circular symmetry) but not the fact that the states are divided into highly degenerate energy levels, called Landau levels, with an energy gap of $\hbar\omega_c$ between levels.

Any physical sample has finite size, and this limits the degeneracy of the Landau levels. We assume a rectangular sample with dimensions L_x, L_y much greater than the magnetic length $l_0 = \sqrt{\hbar/eB}$ and apply periodic boundary conditions in the x direction. This gives

$$k_x = \frac{2\pi m}{L_x} \qquad \text{for integer } m,$$

which fixes the spacing $\Delta y_0 = 2\pi\hbar/eBL_x$ between states in the y direction. Insisting that the center y_0 of each state remain within the sample gives a total of

$$N \equiv \frac{L_y}{\Delta y_0} = \frac{eBL_xL_y}{h} = \frac{L_xL_y}{l_0^2} \gg 1$$

states per Landau level. The degeneracy per area d for each Landau level is

$$d \equiv \frac{N}{L_xL_y} = \frac{eB}{h} = \frac{B}{\Phi_0}, \tag{1.5}$$

where $\Phi_0 = h/e$ is the quantum of flux. Each Landau level has a degeneracy of one electron per flux quantum. Hence, for fixed electron density n, if we vary B then the number of Landau levels which are occupied will change (since the degeneracy changes). We assume that temperatures are low enough that, to good approximation, the lowest energy states are filled first. The occupancy of the Landau levels is given by the filling factor

$$\nu \equiv \frac{n}{B/\Phi_0} \tag{1.6}$$

which is equal to the total number of filled Landau levels (plus the fractional filling of the uppermost Landau level if it is only partially full).

In order to calculate the Hall resistance we modify the Hamiltonian to include a uniform transverse electric field $E_y = E$:

$$H = \frac{1}{2m^*}\left(p_y^2 + (p_x - eBy)^2 + p_z^2\right) + eEy + V(z). \tag{1.7}$$

As before, the z direction separates out and the eigenstates are plane waves in the x direction. The strategy again is to rewrite the Hamiltonian to be a harmonic oscillator shifted by some amount in the y direction. Using $y_1 = -eE/m^*\omega_c^2$, we find

$$\begin{aligned} H_{xy} &= \frac{p_y^2}{2m^*} + \frac{m^*\omega_c^2}{2}(y - y_0)^2 + eEy \\ &= \frac{p_y^2}{2m^*} + \frac{m^*\omega_c^2}{2}\left(y^2 - 2yy_0 + y_0^2 - 2yy_1 + 2y_0y_1 + y_1^2\right) \\ &\quad - m^*\omega_c^2 y_0 y_1 - \frac{m^*\omega_c^2}{2}y_1^2 \\ &= \frac{p_y^2}{2m^*} + \frac{m^*\omega_c^2}{2}(y - y_0 - y_1)^2 + eE(y_0 - y_1) + \frac{e^2E^2}{2m^*\omega_c^2}. \end{aligned} \tag{1.8}$$

Including a normalization factor and ignoring boundary effects in the y direction, the eigenstates are simply

$$\Psi_{j,k_x}(x, y) = \frac{1}{\sqrt{L_x}} e^{ik_x x}\phi_j(y - y_0 - y_1) \tag{1.9}$$

with energies

$$E_{j,k_x} = \hbar\omega_c\left(j + \frac{1}{2}\right) + eE(y_0 + y_1) + \frac{m^*}{2}\left(\frac{E}{B}\right)^2. \tag{1.10}$$

The first component of the energy is the Landau level energy. The second component indicates that the degeneracy is split by the electric field by an amount depending on the center $(y_0 + y_1)$ of the wave function in the y direction. The third component is equal to the classical kinetic energy (with $v = E/B$).

We have so far ignored the confining potential in the y direction. Including this potential would change the wave functions and energies of the eigenstates. However, ϕ_j has a Gaussian envelope with characteristic width l_0, so states centered much further than this from the edge will not be significantly changed by the confining potential. Since the dimensions of the sample are much larger than the magnetic

length, the majority of the states will be essentially unchanged and we are justified in ignoring the confining potential.

Having imposed an electric field in the y direction, we expect from the Hall effect to find a current in the x direction. The current density $\mathbf{J}$ of a single eigenstate is

$$\mathbf{J} = -e\left\{\frac{\hbar}{2im^*}(\Psi^*\nabla\Psi - (\nabla\Psi^*)\Psi) + \frac{e}{m^*}\mathbf{A}|\Psi|^2\right\}. \tag{1.11}$$

Evaluating for the x and y directions gives

$$J_x = -\frac{e\omega_c}{L_x}|\phi_j(y - y_0 - y_1)|^2(y - y_0) \tag{1.12a}$$

$$J_y = 0. \tag{1.12b}$$

To calculate the current I_{j,k_x} due to a single eigenstate we integrate the current density over the length of the sample in the y direction. Because only a small fraction of the states are modified by the confining potential we approximate the limits of integration as $\pm\infty$. Hence,

$$\begin{aligned} I_{j,k_x} &\approx -\frac{e\omega_c}{L_x}\int_{-\infty}^{\infty}(y - y_0)\phi_j^2(y - y_0 - y_1)\,dy \\ &\approx -\frac{e\omega_c}{L_x}\int_{-\infty}^{\infty}(Y + y_1)\phi_j^2(Y)\,dY \\ &\approx -\frac{e\omega_c y_1}{L_x} \\ &\approx -\frac{eE}{L_x B}. \end{aligned} \tag{1.13}$$

The current contributed by each electron eigenstate is independent of j and k_x. The

total current I at a given filling factor scales with the number of occupied states:

$$\begin{aligned} I &= \nu N I_{j,k_x} \\ &= \nu \frac{eBL_xL_y}{h}\left(-\frac{eE}{L_xB}\right) \\ &= -\frac{\nu e^2 V_H}{h} = -\frac{enV_H}{B}. \end{aligned} \tag{1.14}$$

Hence we recover the classical result, Equation (1.1).

The conclusion seems to be that quantum mechanics has no effect on the Hall resistance. However, this is found experimentally to be not the case. In 1980, Klaus von Klitzing et al. discovered that R_H exhibits plateaus as n (or equivalently B) is varied and, strikingly, these plateaus are extremely well quantized [15]. It is easier to quantitatively derive these results using the edge state picture, which we will discuss in Section 1.5. Hence we will content ourselves here with a general argument to explain this behavior. The key is that quantum mechanics does fundamentally change the nature of the system through the creation of Landau levels but the consequences of this are not captured in the ideal system considered above.

Any physical system has impurities and other defects throughout the sample. The presence of these defects broadens the Landau levels (which remain comprised of extended states) and also introduces localized states with a wide energy spectrum. Hence, the Fermi energy may lie not just in the Landau levels, but also between them. In the former case, the picture is essentially the same as described above: R_H scales with B as the number of occupied extended states changes. However, when the Fermi energy lies between Landau levels, changing B only changes the occupancy of localized states. The filled Landau levels are incompressible, unable to accept any electrons, and the total current does not change. Hence R_H is constant while the

Fermi energy lies between Landau levels and, moreover, is quantized at the value determined by the number of filled Landau levels f:

$$R_H = \frac{1}{f}\frac{h}{e^2}. \tag{1.15}$$

Overall, R_H still increases with B, but the linear dependence found above is only accurate on average and is interrupted by plateaus at each integer filling factor.

An experimental observation of the quantized plateaus in R_H is shown as an example in Figure 1.1. Over the same magnetic fields at which R_H is quantized, the longitudinal resistance vanishes. This is also a feature of the quantum Hall effect. Because, at these fields, the filled Landau levels are incompressible, conduction occurs along the sample without dissipation (except for the contact resistance at the source and drain). Hence, in a four-probe configuration the longitudinal resistance is measured to be zero. Again, a more concrete explanation will be discussed in the edge state picture of Section 1.5.

Our entire analysis so far has neglected the spin of the electrons. Luckily, this is easy to remedy, as we can generally neglect spin-orbit coupling and the high magnetic field splits the Landau levels into independent spin-up and spin-down versions. Each of these spin-split Landau levels has the same degeneracy calculated above and contribute equally to R_H. The energy gap between spin-split Landau levels alternates between the Zeeman splitting $g^*\mu_B B$ (approximately $23.2\,\mu$eV/T in GaAs) and the cyclotron energy $\hbar\omega_c$ minus the Zeeman splitting ($\hbar\omega_c \approx 1.73\,$meV/T in GaAs). These values for the energy gaps are modified by various effects, such as electron interactions [17]; however, as long as the thermal excitation remains much less than the energy gaps, the QHE features discussed above will be observed. The experiments reported

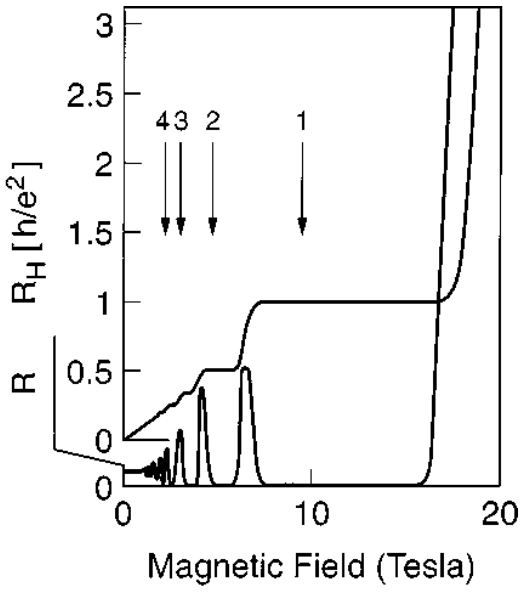


Figure 1.1: **The integer quantum Hall effect.** The Hall resistance R_H and longitudinal resistance R measured as a function of magnetic field in a GaAs/AlGaAs heterostructure. Arrows indicate locations of integer filling factors. Over some range in magnetic field surrounding each of these points R_H exhibits a well-quantized plateau and R vanishes. Figure reproduced from Reference [16].

in this thesis were performed at magnetic fields large enough that the Landau levels are fully spin split. Henceforth we will refer to each spin-split Landau level as just a "Landau level" and will count each of them separately when calculating the filling factor ν and other quantities. For example, $\nu = 2$ indicates the lowest spin-up and spin-down Landau levels are full, or just "two full Landau levels".

The other main difference between the theory presented in this section and actual experiments is the temperature of the system. The derivation above assumed zero temperature, while common experimental systems can only reach electron temperatures on the order of 10 mK. These temperatures are at least two orders of magnitude less than the energy gap at typical magnetic fields of a few Tesla. Practically, the effects of thermal excitation at these temperatures are minimal and the zero-temperature picture developed in this section applies without modification. At higher temperatures, deviations can be observed; in fact, the energy gap is commonly measured by fitting the thermal activation of the longitudinal resistance to an Arrhenius form.

1.4 Fractional Quantum Hall Effect

In 1982, soon after von Klitzing's discovery of the integer QHE, Daniel Tsui and Horst Störmer found similar features at $\nu = 1/3$. Their data is shown in Figure 1.2. Since then, the fractional QHE has been observed at a host of fractional filling factors and, like the integer QHE, is characterized by quantized Hall resistance and vanishing longitudinal resistance. Similar to Equation (1.15), the Hall resistance is quantized

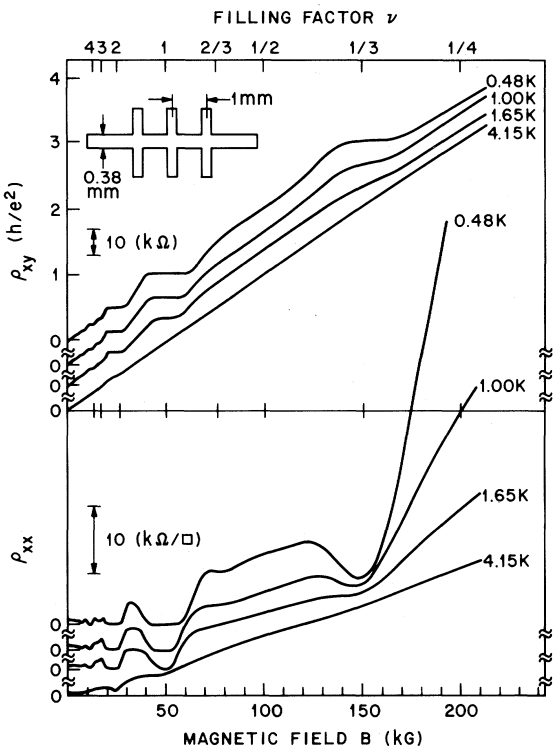


Figure 1.2: **Discovery of the fractional quantum Hall effect.** The first observation of the fractional QHE, at filling factor $\nu = 1/3$, by Tsui et al. [18]. As in the integer QHE (appearing in the plot at $B \lesssim 50\,\mathrm{kG}$), the Hall resistivity ρ_{xy} exhibits a quantized plateau and the longitudinal resistivity ρ_{xx} vanishes. Although these features are not fully developed at the temperatures reported by Tsui et al., the qualitative similarities are clear.

at values

$$R_H = \frac{1}{f}\frac{h}{e^2}, \tag{1.16}$$

where f now takes fractional values, reflecting the number of filled Landau levels, plus the fractional filling of the uppermost Landau level. In many ways, the fractional QHE arises from the same physics as the integer version: the system is incompressible when the Fermi energy lies in a gap between bands of extended states. The differences lie in the details of the cause of the gap and the particles which are carried by the extended states.

The first fractional quantum Hall states to be discovered, and the first to be explained, were those of the Laughlin series. These states occur at filling factors of the form $\nu = 1/(2+q)$ for positive odd integers q. In 1983, Robert Laughlin proposed a variational wave function for these states and explored some of their properties [19]. The many-body Hamiltonian, using complex coordinates $z = x + iy$, is

$$H_{xy} = \sum_k \left\{ \frac{1}{2m^*} \left| \frac{\hbar}{i}\nabla_k + e\mathbf{A}_k \right|^2 + V(z_k) \right\} + \frac{1}{4\pi\epsilon}\sum_{k>l}\frac{e^2}{|z_k - z_l|}, \tag{1.17}$$

where k and l index the electrons and V is a uniform background of an equal amount of positive charge. Laughlin's trial wave function is

$$\Psi_q = \left\{ \prod_{k>l}(z_k - z_l)^{2+q} \right\} \exp\left[-\frac{1}{4l_0^2}\sum_k |z_k|^2 \right]. \tag{1.18}$$

The wave function must be odd under exchange of electrons, requiring that the exponent $(2+q)$ be an odd integer. Laughlin found that this variational wave function has very good overlap with the exact ground state of a system with angular momentum $2+q$ per electron, corresponding to a filling factor $\nu = 1/(2+q)$.

A minimal excitation of this system can be found by adiabatically threading one flux quantum through the xy-plane. The change to the Hamiltonian can be removed by a gauge transformation. However, the wave function will now be in some excited state. Laughlin analyzed the resulting wave function directly by observing that $|\Psi|^2$ has the form of a probability distribution function for a classical one-component plasma (charged particles in a uniform neutralizing background). He found that the excitation caused by threading a flux quantum is the addition or removal of a quasiparticle of charge $e/(2+q)$. We follow a different argument to reach the same result.

Consider a two-dimensional sample with a Corbino disk geometry (an annulus with radii $r_1 < r_2$). We adiabatically thread a flux quantum through the center of the annulus and calculate the total charge that flows across a circle of radius $r_1 < r < r_2$. By Faraday's law, the electric field integrated along the loop equals the rate of change of flux through the loop. Because of the Hall effect, this electric field generates a perpendicular current flowing across the loop (although we presented a global derivation in Sections 1.2 and 1.3, the local current density and electric field vectors obey a similar relation). We integrate over time to calculate the total charge Q that accumulates within the loop:

$$\oint\!\!\!\!\!\!\int \mathbf{E} \cdot \mathbf{dl}\, dt = -\iint \frac{\partial \mathbf{B}}{\partial t} \cdot \mathbf{da}\, dt \tag{1.19}$$

$$\frac{B}{en} \oint\!\!\!\!\!\!\int J_r\, dl\, dt = -\Phi_0$$

$$\frac{B}{en}(-Q) = -\Phi_0$$

$$Q = e\nu. \tag{1.20}$$

Hence the addition of one flux quantum (increasing B) creates a charged excitation in the system with positive charge $e\nu$. Threading a flux in the opposite direction instead introduces a negative charge excitation $-e\nu$. Technically, these excitations may not be *elementary* excitations of the system; that is, Q may be some integer multiple of the charge introduced by an elementary excitation. However, for the Laughlin series, Q is indeed the elementary charge; this has been confirmed by various experiments [6, 20, 21]. These elementary charged excitations of the system can be thought of as quasiparticles carrying the corresponding charge. For fractional filling factors (remember $\nu = 1/(2+q)$ with q odd), the quasiparticle charge is a fraction of the electron charge. Adding or removing charged quasiparticles results in discrete changes to the energy of the system, due to Coulomb interactions. Hence, just like in the integer QHE, the system has an energy gap and becomes incompressible when the Fermi energy lies in that gap. This leads to quantized Hall resistance and vanishing longitudinal resistance. As a final comment, we might ask where the accumulated charge comes from. As long as the radius r of the loop remains constrained to be within the sample ($r_1 < r < r_2$), the conclusion that charge Q is transferred inward remains valid. Hence, the charge must be removed from the outer edge and accumulate on the inner edge of the sample. This is allowed because the gap must close at some point near the edge of the sample as the confining potential increases. The importance of these 'edge states' will be discussed in more detail in Section 1.5.

It turns out, however, that fractional quantum Hall states are not limited to the filling factors $\nu = 1/(2+q)$ given by the Laughlin series. With one important exception, which we will discuss later in this section, the fractional QHE has been

observed at a multitude of fractional filling factors with odd denominators. Figure 1.3 illustrates a number of these states. In response to these experimental observations, a number of people explored possible hierarchies of quantum Hall states to explain the existence of fractional states outside the Laughlin series [22–25]. We will base our discussion on the work of Jainendra Jain, which explains the hierarchy as a quantum Hall effect experienced by composite fermions [25, 26].

To illustrate the concept of composite fermions we imagine a system at filling factor $\nu = 1/2$. This system contains two flux quanta per electron. Energetically, it is favorable for the electrons to be spatially separated because of Coulomb repulsion. A single-particle wave function that achieves this is one with zeros in the wave function at the position of every other electron. This is achieved by putting a vortex in the wave function at each of these positions. These vortices are generated by the flux quanta passing through the system, so we identify the flux quantum with the electron to create a composite particle. At $\nu = 1/2$ the composite particles are each comprised of two flux quanta (or vortices) and one electron. In order to determine the statistics of the composite particle, imagine exchanging two of them. Since electrons are fermions, exchanging them generates a phase factor or $e^{i\pi} = -1$ in the wave function. The phase will also change as each electron circles the flux attached to the other, as in the Aharonov-Bohm effect. An electron completely circling a flux quantum changes in phase by 2π. Exchanging the composite particles produces half of this phase (since two exchanges is the same as a complete circling). Hence, each flux quanta also contributes $e^{i\pi} = -1$ to the phase factor; essentially flux quanta behave like fermions when forming composite particles. With two flux quanta and

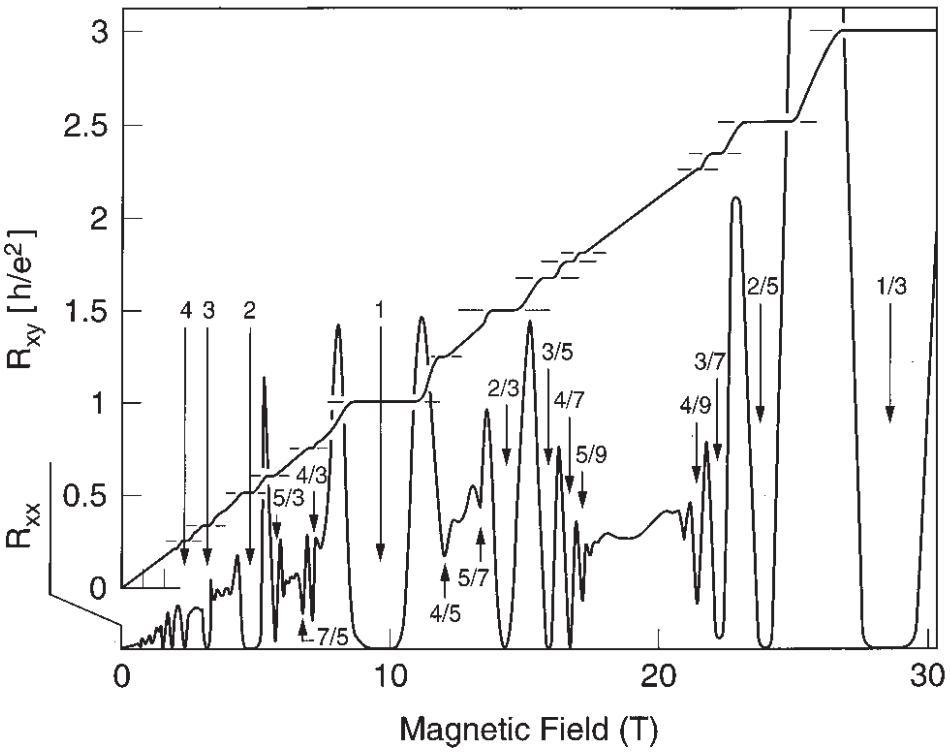


Figure 1.3: **Various fractional quantum Hall states.** Example of many of the observed fractional quantum Hall states in the first two Landau levels. At fractional filling factors with odd denominators, the Hall resistance R_{xy} is quantized and the longitudinal resistance R_{xx} vanishes. Figure reproduced from [27].

one electron, the composite particles at $\nu = 1/2$ are fermions.

Just like electrons, the composite fermions (CFs) undergo their own version of the QHE. At $\nu = 1/2$ all flux quanta have been absorbed into the CFs, which hence experience an effective magnetic field $B_{\text{eff}} = 0$. Adding one flux quantum per CF leads to a single filled CF Landau level. This corresponds to an (electron) filling factor of $\nu = 1/3$. The second CF Landau level occurs when the system contains one-half a flux quantum per CF, corresponding to $\nu = 2/5$. This pattern continues, generating a number of odd-denominator fractional quantum Hall states.

Composite fermions are not only formed at $\nu = 1/2$. CFs comprised of four flux quanta and one electron form at $\nu = 1/4$ and create their own CF Landau levels at $\nu = 1/5, 2/9, \ldots$. Again, the pattern continues indefinitely. Other fractional quantum Hall states can be derived by particle-hole conjugation, which views the Landau level as being formed by holes instead of electrons. In this manner, the $1 - \nu$ state is the conjugate of the ν state (for $\nu \leq 1$). Finally, completely filled lower Landau levels can be inserted without changing the incompressibility of the fractional state. For example, the $\nu = 4/3$ state is, roughly, a $\nu = 1/3$ state with a filled $\nu = 1$ Landau level underneath. To be precise, we will see later in this section that interactions between Landau levels can influence the stability of various fractional states; but for the purposes of the Jain hierarchy, these interactions are ignored.

By applying each of the operations just described in sequence to the $\nu = 1$ integer quantum Hall state, all known integer and fractional states can be derived (with one exception we discuss later). In 1992, Jain and Goldman formalized these operations

as follows [26]:

$$\mathcal{D}: \ \nu = \frac{p}{q} \to \nu' = \frac{\nu}{2\nu + 1} = \frac{p}{2p + q} \qquad \text{(CF Landau levels)} \tag{1.21a}$$

$$\mathcal{C}: \ \nu = \frac{p}{q} \to \nu' = 1 - \nu = \frac{q - p}{q} \qquad \text{(particle-hole conjugation)} \tag{1.21b}$$

$$\mathcal{L}: \ \nu = \frac{p}{q} \to \nu' = 1 + \nu = \frac{p + q}{q} \qquad \text{(adding a filled Landau level),} \tag{1.21c}$$

for integer p and odd integer q. They proved that each ordered combination of these operators, applied to $\nu = 1$, produces a unique state and that all odd-denominator filling factors (including the integers with $\nu = p/1$) can be obtained by such combinations. Essentially the entirety of the integer and fractional QHEs can be understood through the resulting hierarchy. Figure 1.4 shows the beginnings of this hierarchy for various combinations of the operators in Equations (1.21).

As we indicated in our discussion above, there is one pair of fractional quantum Hall states which appear outside of the Jain hierarchy. In 1987, Willett et al. observed a fractional quantum Hall state at $\nu = 5/2$ [28], and one at $\nu = 7/2$ (related to $\nu = 5/2$ by particle-hole symmetry) has also been observed [29]. No fractional quantum Hall states have been observed at other half-integer fillings, such as $\nu = 1/2$, at least in single-layer electron systems. Systems with either a wide quantum well or two closely spaced quantum wells (a "double quantum well") have been observed to exhibit a $\nu = 1/2$ fractional quantum Hall state [30, 31], as well as other even-denominator states; however, in this thesis we focus exclusively on single-layer systems. With regard to these systems, two questions naturally arise. What leads to the fractional QHE at $\nu = 5/2$ and $7/2$? Why don't similar states arise at other half-integer filling factors?

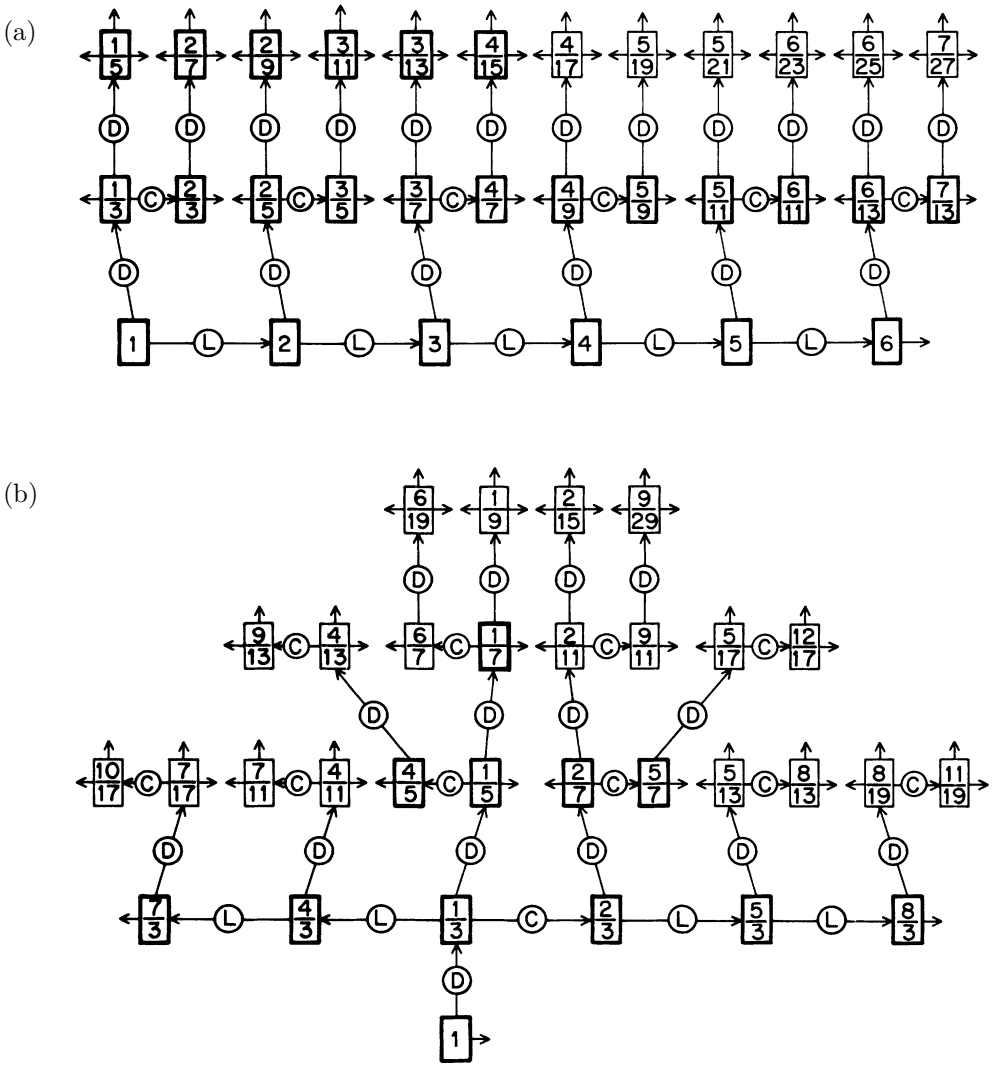


Figure 1.4: **The Jain hierarchy.** Beginnings of the Jain composite fermion hierarchy. Boxed numbers are filling factors and labeled arrows indicate the use of operators to derive additional states. The operators $\mathcal{D}$, $\mathcal{C}$, and $\mathcal{L}$ are defined in Equations (1.21). Figures reproduced from Reference [26].

A number of different wave functions describing the $\nu = 5/2$ state have been proposed. We discuss these in more detail in Section 3.2. However, all these states share a common feature—the pairing of composite fermions—to explain the existence of a fractional quantum Hall state at an even-denominator filling factor. As discussed above, at $\nu = 1/2$ CFs form as a combination of two flux quanta and one electron. Similar to Cooper pairs in BCS superconductivity, the pairing of CFs reduces the total energy of the system and opens an energy gap. Because the paired CFs have double the charge of a single CF the corresponding flux quantum of the system (the amount of flux which can be removed by a gauge transformation) changes to $\Phi_0' = h/2e$. This modifies the results of Equation (1.20) such that the quasiparticle charge must divide $Q' = e\nu/2 = e/4$. As we will see in Section 3.2, the quasiparticle charge at $\nu = 5/2$ has been measured to be $e/4$.

So far we haven't discussed the statistics of the quasiparticles in the various fractional quantum Hall states. It turns out that quasiparticles arising in fractional quantum Hall states in the Jain series are anyons, meaning that they acquire an arbitrary phase factor of $e^{i\theta}$ (determined by the specific state) upon particle exchange [3, 32]. This type of particle statistics can only occur in two-dimensional systems. In three dimensions the path of one particle circling another can always be continuously deformed to a point and hence the wave function must remain unchanged under such an operation. Exchanging two particles introduces a phase factor of $e^{0i} = 1$ or $e^{i\pi} = -1$; these are the well-known bosons and fermions. In two dimensions, such a path deformation is not allowed (since it involves deforming the path into the third dimension) and the accumulated phase in certain systems can take any value, hence

the term 'anyon'.

The $\nu = 5/2$ state may actually exhibit even more exotic particle statistics. A number of the proposed descriptions of this state posses non-abelian statistics [4]. In this case exchanging quasiparticles generates a unitary transformation of the wave function. Because this type of matrix operation is not commutative, the final wave function is dependent on the order of quasiparticle exchanges and the statistics is termed 'non-abelian'. The set of wave functions which can be obtained by such unitary transformations form a highly degenerate space of size $\sim 2^m$ for a system of $2m$ quasiparticles [33]. Such non-abelian statistics are inherently interesting, especially if they can be observed and manipulated in a physical system. In addition, proposals for topological quantum computers, which may be more robust against environmental decoherence, rely on non-abelian statistics [34].

We return to our second question about the half-integer fractional quantum Hall states, namely why they are not observed at filling factors other than 5/2 and 7/2. Whether the state at half-integer filling is compressible or incompressible (with an energy gap) depends on the details of the interactions of electrons in that Landau level. Haldane has shown that the interactions can be completely described by a set of pseudopotential coefficients which are equal to the energies of two-particle states of various relative angular momenta [35]. Because of differences in the shape of the electron wave function, the values of these pseudopotentials are dependent on the Landau level. The differences in the pseudopotentials for $\nu = 5/2$ compared to $\nu = 1/2$ have been found to be favorable for the formation of an incompressible fractional quantum Hall state [36]. In addition, mixing between the Landau levels has

been observed [37] and can affect the stability of quantum Hall states by reducing the energy gap [38]. In particular, Landau level mixing is believed to modify the energy gap at $\nu = 5/2$ and may influence whether the state exhibits non-abelian statistics [39].

1.5 Edge State Picture

The role of the edges of a two-dimensional sample was considered in detail by Bertrand Halperin in 1982 [40]. His fundamental insight was that, while the filled Landau levels lie below the Fermi energy throughout the sample, they must actually cross the Fermi energy near the boundary of the sample. Because of the confinement potential, the energy of each Landau level must increase as it approaches the boundary, eventually crossing the Fermi energy. Lower-lying Landau levels (smaller ν) cross closer to the boundary. At the points where a Landau level crosses the Fermi energy, the gap closes and a one-dimensional state with a continuum of excitations forms. These 'edge states' are extended and carry current around the boundary of the sample. The strong magnetic field forces the edge states to be chiral—edge states carrying electrons to the left are always on the opposite side of the sample as edge states carrying electrons to the right. Halperin developed this theory of edge states and showed that they exist even in the presence of disorder [40].

The fundamental features of the integer QHE are nearly trivial consequences of the edge state picture. Each edge state is a one-dimensional channel and hence has one quantum of conductance e^2/h arising from its contact with electron reservoirs at the source and drain. For f filled Landau levels, the Hall resistance is equal to the

potential difference between these reservoirs divided by the total current carried by the f edge states:

$$R_H = \frac{1}{f}\frac{h}{e^2},$$

which is the same result as Equation (1.15). The longitudinal resistance, in contrast, probes the dissipation of the edge states as they travel between the reservoirs. The edge states are confined to be within a few magnetic lengths of the boundary of the sample. For the macroscopic samples that are used in experiments, this suppresses the tunneling between edge states on opposite sides of the sample to be effectively zero. Since all the edge states on one side carry electrons in the same direction, there is no backscattering of electrons and hence no dissipation of current. The longitudinal resistance is zero, as expected.

This picture is easily extended to the fractional QHE by allowing fractional edge states having conductance equal to a fraction of e^2/h. The Laughlin series states, for example, each have conductance $\nu e^2/h$. Complications can arise when the edge states interact with each other or the confining potential. This can lead to reconstruction of edge states such that form of the edge states is not the same as that described by this simple picture [41]. For example, in this picture the $\nu = 2/3$ state is thought of as having a $\nu = 1$ edge of electrons traveling downstream (the usual direction) and a $\nu = -1/3$ edge of holes traveling upstream (because they have opposite charge) [42, 43]. However, reconstruction is expected to change these edge states to a $\nu = 2/3$ downstream edge and a neutral upstream edge [44]; this arrangement is supported by experimental evidence [12, 13].

Because the edge states are single-mode ballistic conductors, the Landauer-Büttiker

formalism [45] provides a powerful tool to analyze systems of various geometries. Assuming that the ohmic contacts completely transmit any incident edge states [46], the current sourced by contact k at chemical potential μ_k is

$$I_k = \frac{e^2}{h}\left(\mu_k N_k^{\text{out}} - \sum_l \mu_l N_l^{\text{in}}\right), \tag{1.22}$$

where the sum is performed over all ohmic contacts. The number of edge states leaving contact k is given by N_k^{out}. The number of edge states incident on contact k coming from each contact l is N_l^{in}. In cases where the transmission of edge states between contacts is not unity, we allow N_l^{in} to be a continuous variable, determined by the appropriate transmission amplitudes between edge states. Counting the number of edge states is really just a method of determining the total conductance; hence fractional edge states contribute fractional amounts to N^{out} and N^{in}.

As an example, we apply the Landauer-Büttiker formalism to the simple Hall bar with a fixed current source I illustrated in Figure 1.5. We write one equation for each ohmic contact, resulting in a system of linear equations. This allows us to solve for the chemical potential at each contact as a function of the number of edge states N_B. The linear equations are:

$$\text{S}: \quad I = \frac{e^2}{h}(\mu_{\text{S}} N_B - \mu_{\text{L1}} N_B) \tag{1.23a}$$

$$\text{L1}: \quad 0 = \frac{e^2}{h}(\mu_{\text{L1}} N_B - \mu_{\text{R1}} N_B) \tag{1.23b}$$

$$\text{L2}: \quad 0 = \frac{e^2}{h}(\mu_{\text{L2}} N_B - \mu_{\text{S}} N_B) \tag{1.23c}$$

$$\text{R1}: \quad 0 = \frac{e^2}{h}(\mu_{\text{R1}} N_B - \mu_{\text{D}} N_B) \tag{1.23d}$$

$$\text{R2}: \quad 0 = \frac{e^2}{h}(\mu_{\text{R2}} N_B - \mu_{\text{L2}} N_B) \tag{1.23e}$$

$$\text{D}: \quad -I = \frac{e^2}{h}(\mu_{\text{D}} N_B - \mu_{\text{R2}} N_B). \tag{1.23f}$$

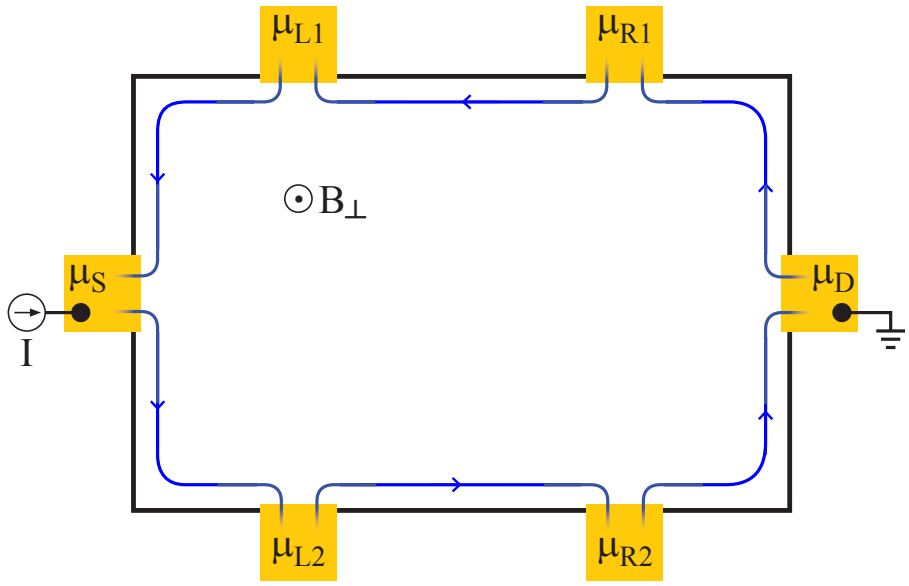


Figure 1.5: **Edge states in a simple Hall bar.** A simple Hall bar (black outline) containing a two-dimensional electron system in the integer or fractional quantum Hall regimes. Ohmic contacts (gold) on the left and right act as a current source and drain. The remaining four contacts are used as voltage probes. All ohmic contacts are labeled according to their chemical potentials μ. The blue line represents one or more edge states, which carry current in the direction of the arrows.

For this simple example solving for the chemical potential at each contact is straightforward. We choose $\mu_{\mathrm{D}} = 0$ and solve for the remaining contacts:

$$\mu_{\mathrm{L1}} = \mu_{\mathrm{R1}} = \mu_{\mathrm{D}} = 0 \tag{1.24a}$$

$$\mu_{\mathrm{S}} = \mu_{\mathrm{L2}} = \mu_{\mathrm{R2}} = \frac{I}{N_B}\frac{h}{e^2}. \tag{1.24b}$$

We calculate the Hall resistance as

$$R_H = \frac{\mu_{\mathrm{L2}} - \mu_{\mathrm{L1}}}{I} = \frac{\mu_{\mathrm{R2}} - \mu_{\mathrm{R1}}}{I} = \frac{1}{N_B}\frac{h}{e^2}. \tag{1.25}$$

This reproduces the result of Equation (1.15), extended to include the fractional QHE. Similarly, the longitudinal resistance is

$$\frac{\mu_{\mathrm{L1}} - \mu_{\mathrm{R1}}}{I} = \frac{\mu_{\mathrm{L2}} - \mu_{\mathrm{R2}}}{I} = 0, \tag{1.26}$$

as expected.

In 1990, Xiao-Gang Wen demonstrated that the excitations in a fractional quantum Hall edge state form a chiral Luttinger liquid (CLL) [2]. He and others have further developed this theory and explored the properties of CLLs since then; for a review of theory and experiment see [47]. Unlike in the integer quantum Hall regime, CLLs describing fractional edge states must account for the interactions between quasiparticles. CLL theory characterizes these interactions by the parameter g describing the interaction strength. Together g and the quasiparticle charge e^* determine the physics of a CLL.

One of the most important results of CLL theory is the form of quasiparticle tunneling into an edge state. Experimentally, tunneling between two edge states can be induced by bringing the opposite edges of the sample close together, for example

by using a QPC. We will use this technique to study tunneling of the $\nu = 5/2$ edge state in Chapter 3. Another common technique is the use of cleaved-edge overgrowth samples, in which tunneling occurs between a fractional quantum Hall edge and a three-dimensional metal separated by a thin barrier [47].

In Chapter 3 we make use of a calculation by Wen of the differential tunneling conductance g_t between two edge states [1]. However, g_t is not a quantity we measure directly in our experimental setup. Hence we want to relate g_t to the voltage measurements at various ohmic contacts, which are quantities we can measure. We begin by applying the Landauer-Büttiker formalism to a Hall bar with a QPC constriction in the middle, as illustrated in Figure 1.6. The QPC brings edge states from opposite sides of the sample into proximity, allowing quasiparticles to tunnel between the edge states. In addition, some number of edge states may be completely reflected by the QPC (not illustrated in the figure). Together, these two effects change the chemical potentials of various contacts from the simple case analyzed above.

As before, we define N_B as the number of edge states in regions far away from the QPC. We assume that a maximum of one edge state is partitioned at the QPC, with all inner edge states (corresponding to larger ν) being completely reflected and all outer edge states (smaller ν) being completely transmitted. Hence we are able to define N_D as the number of fully transmitted edge states plus the transmission probability of the partitioned edge state times its conductance. In other words, N_D is the number of edge states passing through the QPC, taking into account the partial transmission of the partitioned edge state. By changing the tunneling probability, N_D can be varied continuously. We modify Equations (1.23) to account for the addition

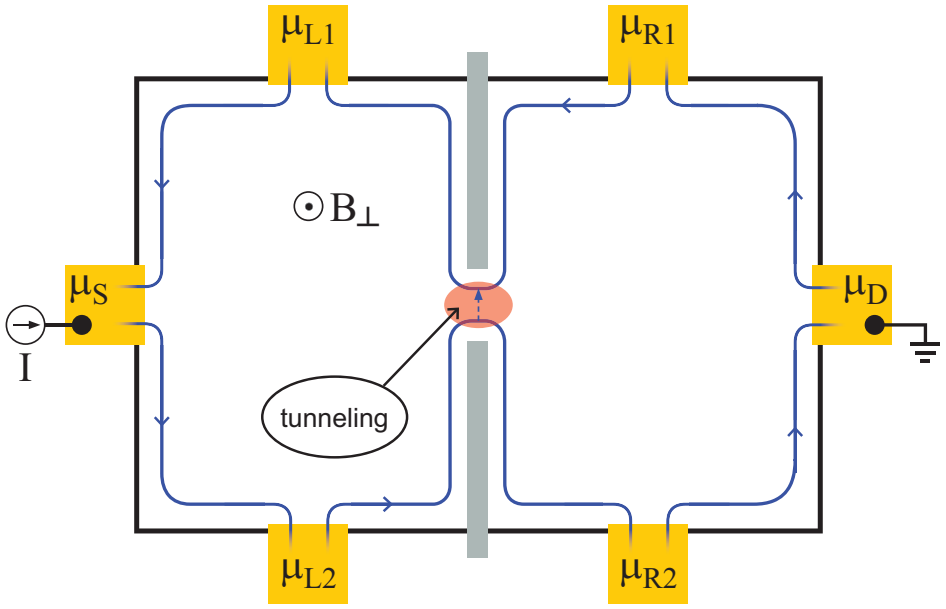


Figure 1.6: **Edge states in a QPC.** A Hall bar (black outline) containing a two dimensional electron system in the integer or fractional quantum Hall regimes. Ohmic contacts (gold) on the left and right act as a current source and drain. The remaining four contacts are used as voltage probes. All ohmic contacts are labeled according to their chemical potentials μ. The blue line represents one or more edge states, which carry current in the direction of the arrows. A quantum point contact (grey) brings the edge states from opposite sides of the sample into proximity (pink region), allowing quasiparticles to tunnel.

of the QPC:

$$\text{S}: \quad I = \frac{e^2}{h}(\mu_{\text{S}} N_B - \mu_{\text{L1}} N_B) \tag{1.27a}$$

$$\text{L1}: \quad 0 = \frac{e^2}{h}(\mu_{\text{L1}} N_B - \mu_{\text{R1}} N_D - \mu_{\text{L2}}(N_B - N_D)) \tag{1.27b}$$

$$\text{L2}: \quad 0 = \frac{e^2}{h}(\mu_{\text{L2}} N_B - \mu_{\text{S}} N_B) \tag{1.27c}$$

$$\text{R1}: \quad 0 = \frac{e^2}{h}(\mu_{\text{R1}} N_B - \mu_{\text{D}} N_B) \tag{1.27d}$$

$$\text{R2}: \quad 0 = \frac{e^2}{h}(\mu_{\text{R2}} N_B - \mu_{\text{L2}} N_D - \mu_{\text{R1}}(N_B - N_D)) \tag{1.27e}$$

$$\text{D}: \quad -I = \frac{e^2}{h}(\mu_{\text{D}} N_B - \mu_{\text{R2}} N_B). \tag{1.27f}$$

As before, we set $\mu_{\text{D}} = 0$ and solve for the other chemical potentials as a function of I, N_B, and N_D:

$$\mu_{\text{R1}} = \mu_{\text{D}} = 0 \tag{1.28a}$$

$$\mu_{\text{R2}} = \frac{I}{N_B}\frac{h}{e^2} \tag{1.28b}$$

$$\begin{aligned} \mu_{\text{S}} = \mu_{\text{L2}} &= \mu_{\text{R2}}\frac{N_B}{N_D} \\ &= \frac{I}{N_D}\frac{h}{e^2} \end{aligned} \tag{1.28c}$$

$$\begin{aligned} \mu_{\text{L1}} &= \mu_{\text{L2}}\frac{N_B - N_D}{N_B} \\ &= I\frac{h}{e^2}\frac{N_B - N_D}{N_B N_D}. \end{aligned} \tag{1.28d}$$

Examining the chemical potentials on either side of the QPC, we see that the QHE remains valid in each of the two halves of the Hall bar. That is

$$R_H = \frac{\mu_{\text{L2}} - \mu_{\text{L1}}}{I} = \frac{\mu_{\text{R2}} - \mu_{\text{R1}}}{I} = \frac{1}{N_B}\frac{h}{e^2}, \tag{1.29}$$

in agreement with Equation (1.25). Additionally, it is easy to see that the chemical potential does not change along any edge far away from the QPC and hence the

longitudinal resistance is zero. Before proceeding, we consider the limit $N_D \to N_B$ in the equations above. This returns us to the simple Hall bar illustrated in Figure 1.5, and Equations (1.28) reduce to Equations (1.24), as expected.

One can also measure between ohmic contacts on opposite sides of the QPC, and these results will reflect the physics of the quasiparticle tunneling. We define the diagonal resistance $R_D^+ = (\mu_{\mathrm{L2}} - \mu_{\mathrm{R1}})/I$. Other resistances can be defined for different pairs of ohmics, but we will only need R_D^+ to determine g_t. The differential tunneling conductance is defined as

$$g_t = \frac{dI_t}{dV_t}, \tag{1.30}$$

for tunneling current I_t and tunneling voltage V_t. We define the current reflected by the QPC to be I_R. Note that in the setup of Figure 1.6 the transmitted current is just I, because the only drain in the system is ohmic contact D. The total current $I_0 = I_R + I$ carried by the N_B edge states incident on the QPC (leaving contact L2) is

$$I_0 = \frac{e^2}{h}\mu_{\mathrm{L2}} N_B = \frac{\mu_{\mathrm{L2}}}{R_H}. \tag{1.31}$$

It may be confusing that $I_0 > I$, but remember that I_R is reflected by the QPC and, because there is no drain for it to enter, ultimately returns to contact S. The *net* current through S (and hence through the entire sample) is just I. At this point we consider only systems in which there are no completely reflected edge states at the QPC, or in other words, the edge state partitioned by the QPC is the innermost one. In this situation we have $I_t = I_R$.

There is some ambiguity as to the proper tunneling voltage to use, $V_H = \mu_{\mathrm{R2}} - \mu_{\mathrm{R1}}$ or $V_D^+ = \mu_{\mathrm{L2}} - \mu_{\mathrm{R1}}$. It turns out that the choice, in this case, is actually irrelevant. Wen's calculation of g_t is a first order perturbation in the tunneling amplitude, and

hence is only valid for weak tunneling. To zeroth order $V_H = V_D^+$ and therefore to calculate the first order of g_t, either can be used for the tunneling voltage. Here we will follow Wen [1] and use $V_t = V_H$.

We are now in a position to calculate g_t as a function of measured quantities. We first note, using Equations (1.28) and (1.29),

$$g_t = \frac{dI_R}{dV_H} = \frac{1}{R_H}\frac{dI_R}{dI}. \tag{1.32}$$

We now manipulate the expression for diagonal resistance to calculate the tunneling conductance. Because we want the differential tunneling conductance, we actually use the differential diagonal resistance:

$$\begin{aligned} r_D^+ &= \frac{dV_D^+}{dI} \qquad (1.33)\\ &= \frac{d\mu_{\mathrm{L2}}}{dI} \\ &= r_H \frac{dI_0}{dI} \\ &= r_H(\frac{dI_R}{dI} + \frac{dI}{dI}) \\ &= r_H(r_H g_t + 1). \qquad (1.34) \end{aligned}$$

We have also replaced R_H with the differential Hall resistance $r_H = dV_H/dI$, since the two are equal when the Landauer-Büttiker formalism is applicable. Rearranging, gives

$$g_t = \frac{r_D^+ - r_H}{r_H^2}. \tag{1.35}$$

Chapter 2

Experimental Methods

2.1 Experimental Details

The measurements presented in Chapters 3 and 4 were performed with a device fabricated on a GaAs/AlGaAs heterostructure. In this section we discuss the characteristics of this heterostructure and present experimental details and methods.

A heterostructure is simply a material formed from layers of various semiconductors—in our case GaAs and AlGaAs. High-quality samples are grown using molecular beam epitaxy (MBE), which allows for precision control of the thickness and composition of each layer. The purpose of using a heterostructure—rather than, say, a crystal of GaAs—is to limit the electrons to a two-dimensional region. As we will discuss, the geometry of this region can be controlled both lithographically during fabrication and with metallic gates during data collection. The two-dimensional confinement is achieved by putting the electrons in a thin layer of GaAs sandwiched between $Al_xGa_{1-x}As$, where x denotes the fraction of Ga atoms replaced by Al. Be-

cause GaAs and AlAs both form zincblende crystal structures with nearly identical lattice constants, the fraction x can be freely controlled without inducing lattice mismatch or strain. However, the band gap does vary with x. The conduction band energy of GaAs is lower than that of $Al_xGa_{1-x}As$ by roughly $0.8x$ eV for $0 \leq x \leq 0.45$ (in which range the band gap is direct) [48]. Hence the electrons are attracted to the GaAs layer and at low temperatures this potential difference forms a quantum well. If temperatures are low enough that the electrons are only able to access the ground state of the well then they form what is effectively a two-dimensional electron system (2DES) in the layer of GaAs. For reference, using a simple finite square well model gives an energy difference of $\sim 15\,\text{meV} \approx 170\,\text{K}$ between the ground and first excited states for the heterostructure reported on in this thesis (quantum well width of 30 nm and $x = 0.24$).

The electrons come from two layers of Si δ-donors spaced equidistant on either side of the quantum well, separated from it by the AlGaAs. As implied by the name, δ-donors are localized to a single atomic plane in the crystal [49]. As long as the density of Si is not too low (underdoped) the electron density in the 2DES is determined by the thickness of the setback of the Si from the GaAs quantum well. The Si donors can be set directly into the AlGaAs or into thin GaAs quantum wells sandwiched by AlGaAs. Another layer of Si donors may be placed closer to the surface of the wafer to help compensate for surface states. In addition, a thin cap layer of GaAs is grown to help reduce the effects of oxidation. The active region of a sample heterostructure with double-sided doping is illustrated in Figure 2.1a. The metallic gates shown on the surface of the chip will be discussed below.

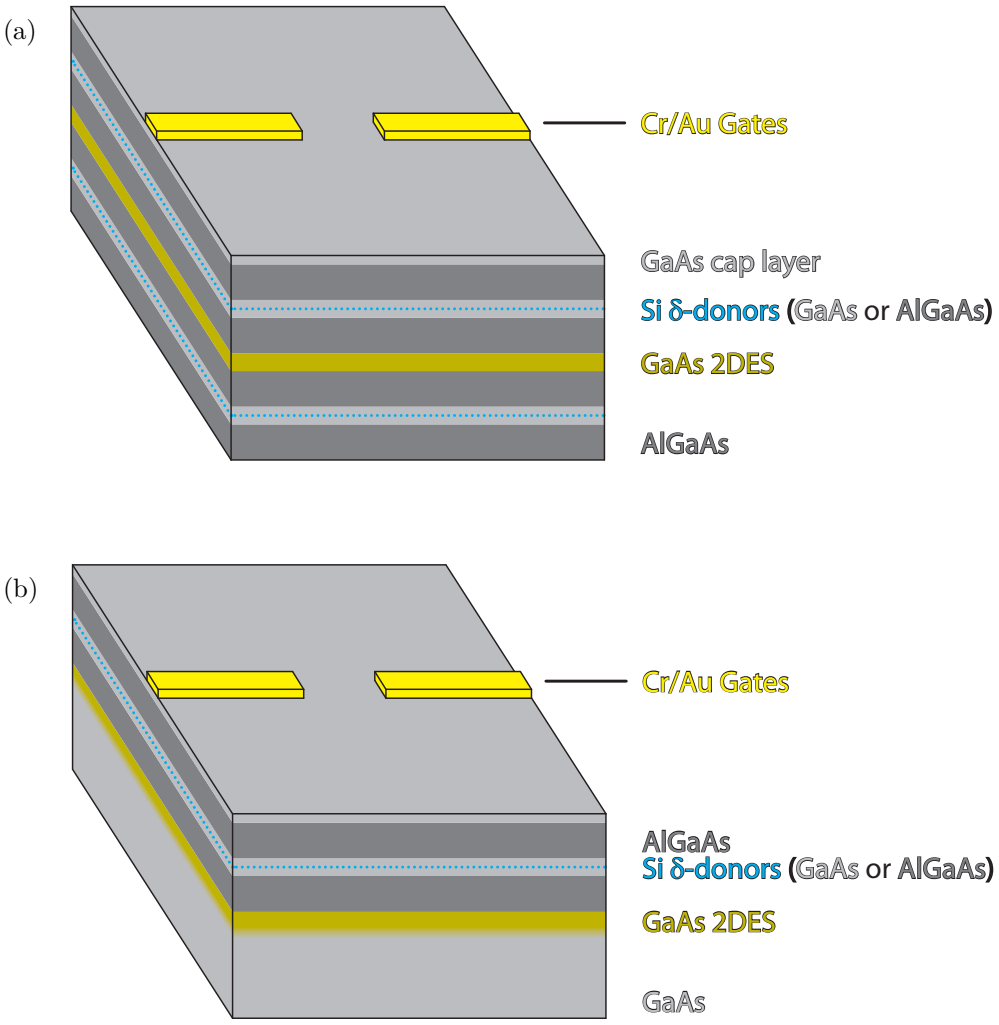


Figure 2.1: **Common GaAs/AlGaAs heterostructures.** Cross-section of two common GaAs/AlGaAs heterostructure patterns (not to scale). (a) A square quantum well with double-sided doping. (b) A triangular quantum well with single sided doping. For both, metallic gates are shown deposited on the surface of the chip (top of diagram). Layers of different materials are illustrated by different colors and labeled on the right of each diagram. The Si δ-donors are often deposited directly into AlGaAs, but may instead have their own thin GaAs quantum well, as illustrated.

Although modern high-mobility heterostructures are usually grown with a square quantum well and two layers of δ-donors, it is also possible to create a single-sided heterostructure, containing only a single interface between GaAs and AlGaAs. In this case only a single layer of δ-donors is used (on the AlGaAs side) and the quantum well has a triangular shape. The side towards the δ-donors has a sharp vertical potential arising from the AlGaAs/GaAs interface. Confinement on the other side is not provided by AlGaAs, but rather by the attractive potential of the ionized Si donors, hence leading to a triangular potential well. A sample heterostructure with single-sided doping is illustrated in Figure 2.1b.

The device discussed in Chapters 3 and 4 is fabricated from a double-sided GaAs/AlGaAs heterostructure with a square quantum well. Details of the fabrication procedure are discussed in Appendix B. The heterostructure wafer was grown by Loren Pfeiffer and Ken West at Bell Labs (they have since moved their MBE facility to Princeton University) and is wafer 2_25_05.1 in their numbering system. The GaAs quantum well is 30 nm thick and is centered 210 nm from the surface of the wafer. Si δ-donors are located 98 nm above and below the center of the GaAs quantum well. The donors are contained in their own $\sim$ 3 nm wide quantum wells. There is no extra layer of Si to compensate for surface states and the uppermost 10 nm of the heterostructure is a GaAs cap layer. The electron density of this material is $2.6 \times 10^{11}\,\mathrm{cm}^{-2}$. The mobility was originally measured to be $2 \times 10^{7}\,\mathrm{cm}^{2}\,\mathrm{V}^{-1}\,\mathrm{s}^{-1}$ [50], but more recently was found to have degraded to $1 \times 10^{7}\,\mathrm{cm}^{2}\,\mathrm{V}^{-1}\,\mathrm{s}^{-1}$. The cause of this degradation is unknown.

A Hall bar containing the device is defined by etching from the surface of the

chip down through the GaAs quantum well layer. This isolates the 2DES into a well-defined region, called a mesa. A diagram of the Hall bar, including the ohmic contacts and metallic surface gates, is shown in Figure 2.2 (left). The Hall bar is 150 μm wide and roughly 2 mm long. After the Hall bar is etched, an alternating stack of Pt/Au/Ge is deposited on areas where contact to the 2DES is desired and then annealed to form ohmic contacts. Metallic gates are then deposited on the surface to give electrostatic control over the device. By applying a negative voltage to any of these gates, the electron density in the 2DES underneath can be reduced and eventually depleted to zero. Applying an even more negative gate voltage increases the size of the depletion region, changing the dimensions and characteristics of the device. This is illustrated in Figure 2.3. In this manner the geometry of the 2DES can be modified to form various devices such as QPCs [50], quantum dots [51, 52], and interferometers [6, 7], which can be controlled in real time to a scale of $\sim$ 10–100 nm. The gate pattern forming the active region of the device is show in Figure 2.2 (right). The lithographic distance between gates A1 and G3 (or similarly A2 and G2) is $\sim 0.6\,\mu$m. The distance between G2 and G3 is $\sim 1.2\,\mu$m and the length of each is $\sim 1.8\,\mu$m.

The chip is wire bonded to a chip carrier and cooled in a ^{3}He/^{4}He dilution refrigerator. The mixing chamber has a base temperature of 3–5 mK. The electron temperature tracks the mixing chamber temperature well down to $\sim$ 20 mK but diverges below this point as the electrons decouple from the lattice. At these lower temperatures the electrons are cooled primarily through the wires attached to the ohmic contacts. Unfortunately, it is hard to make good thermal contact between these coaxial cables and the mixing chamber. The electron temperature below 20 mK has been calibrated

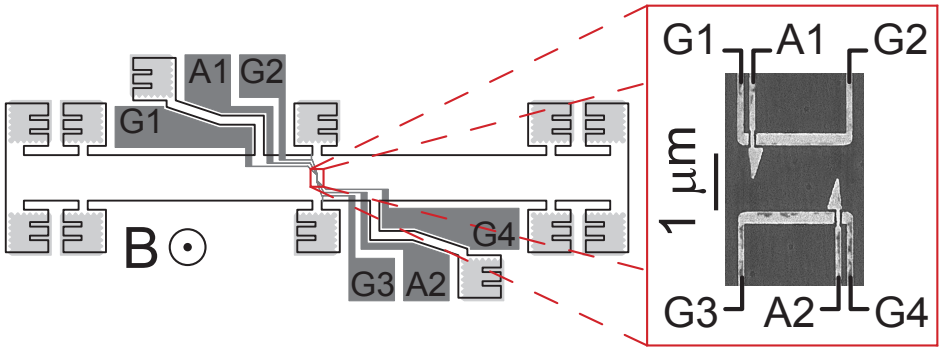


Figure 2.2: **Diagram of experimental Hall bar and SEM image of gate geometry.** (left) Diagram of the Hall bar used for experiments reported in this thesis. The 2DES exists in the mesa outlined in black. The magnetic field B is perpendicular to the plane of the 2DES. Light grey regions indicate ohmic contacts and dark grey regions are metallic gates deposited on the surface of the chip. The active region of the device is in the center of the Hall bar, where the gates converge, and is shown in detail in the box. (right) A scanning electron micrograph of a device fabricated in a similar manner to the one reported on in this thesis. The surface of the chip is dark grey and the metallic gates are light grey. Gate labels are included in both parts.

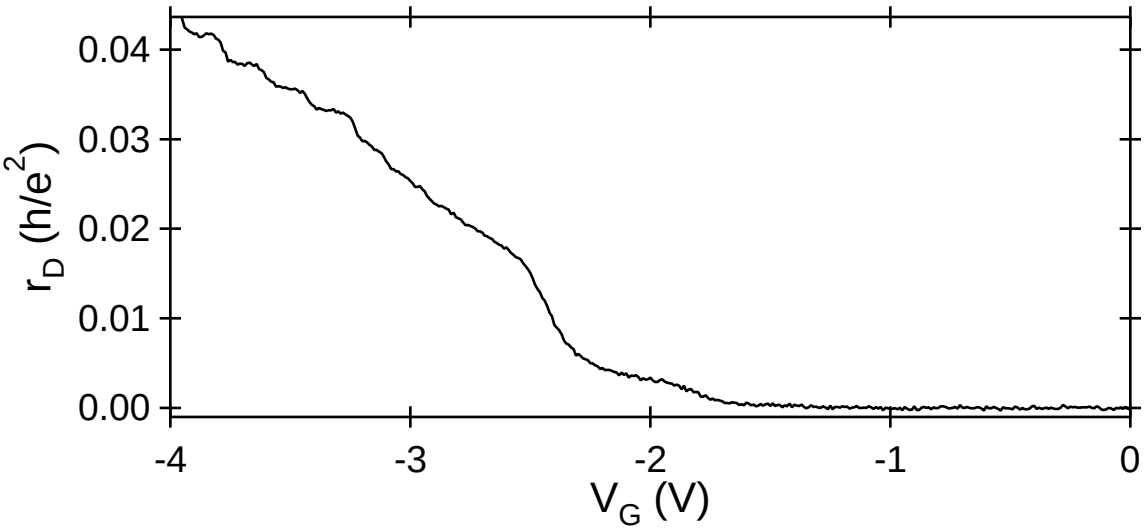


Figure 2.3: **Depletion and pinch off of a QPC with gate voltage.** Differential resistance r_D (defined below) through a QPC as a function of gate voltage V_G. As the gates are energized, the electron density underneath is reduced but the resistance remains nearly zero until the gates begin to deplete the underlying 2DES at roughly $-1.7\,\mathrm{V}$. By approximately $-2.5\,\mathrm{V}$ the gates are fully depleted and the QPC has formed. The nonlinearity in this range is believed to be a result of the gate width decreasing by roughly two orders of magnitude, from the wide bonding pads to the smallest features defining the QPC. The electrons underneath the wider regions of the gates deplete at more positive gate voltages, leading to the step observed at $-2\,\mathrm{V}$. Once the gate voltage has been reduced past $-2.5\,\mathrm{V}$, the QPC is completely formed and more negative voltage serves to reduce its width. In this regime the resistance increases roughly linearly with more negative gate voltage. For voltages more negative than $-3\,\mathrm{V}$ a step-like structure results as the number quantum electrostatic subbands passing through the QPC is reduced. This effect is discussed in more detail in Section 2.3. Although not shown in this graph, applying large enough negative voltage will completely pinch off the QPC and the resistance will increase without bound. The QPC measured here is formed with gates A1 and G3 (see Figure 2.2).

with thermal broadening of Coulomb blockade peaks and by extrapolating features of the QHE found to be linear in temperature [53] (also see Supporting Online Material of [50]). The electron temperature is found to be $\sim 15\,\mathrm{mK}$ when the mixing chamber is at base temperature. The sample is placed in the center of a superconducting coil magnet which supplies the perpendicular magnetic field needed to achieve the QHE. The magnetic field can be held constant with very low drift because the supercurrent decays very slowly. Alternatively, by heating a small length of the wire above the superconducting temperature, the current can be shunted through the external power supply. This allows the current, and hence the magnetic field, to be swept in a continuous manner.

The measurement setup is illustrated in Figure 2.4. A current is sourced through ohmic contacts on one end of the Hall bar and drained through contacts on the other end. The AC excitation is fixed to 0.4 nA RMS at 17 Hz and the DC bias is varied as an independent variable. Voltages can be measured at the remaining contacts. In order to minimize the measurement noise we use lock-in amplifiers to pick out the 17 Hz component of the voltage response. Dividing this by the AC current amplitude results in a measurement of differential resistance. Common measurements we perform are the differential Hall resistance $r_{XY} = r_H$ and differential diagonal resistance r_D as illustrated in Figure 2.4. The diagonal resistance r_D is effectively the same as r_D^+ discussed in Section 1.5 when $r_{XX} = 0$. In particular, Equation 1.35 is still valid with r_D in place of r_D^+. However, r_D has the added advantage that, because the contacts are positioned directly opposite each other across the Hall bar, it has no contribution from the longitudinal resistance. Hence, even when the sample is not in

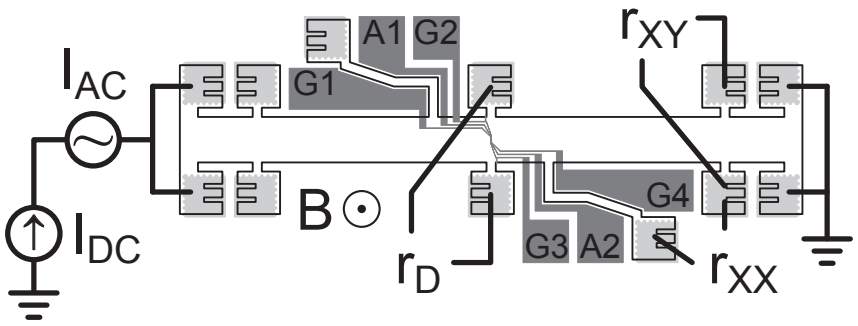


Figure 2.4: **Diagram of experimental Hall bar with illustration of electrical measurements performed.** A current with AC and DC components is sourced at ohmic contacts on the left of the Hall bar and drained through grounded contacts on the right. Differential resistance measurements of the Hall resistance r_{XY} and diagonal resistance r_D are performed. The differential longitudinal resistance r_{XX} can also be measured as illustrated if gate G4 is grounded (otherwise the gate cuts off the two contacts from each other).

the quantum Hall regime r_D reflects the local Hall resistance in the QPC, while r_D^+ also contains a contribution from the longitudinal resistance. The differential longitudinal resistance r_{XX} illustrated in Figure 2.4 can only be measured when the gate G4 is grounded; however, this is indeed the case for many of the device configurations for which we present results. We note that the Landauer-Büttiker formalism results in $r_{XY} = R_{XY}$ and $r_{XX} = R_{XX}$ when the sample is in the quantum Hall regime.

2.2 Gate Annealing Technique

As illustrated in Figure 2.3, applying a negative voltage to the gates depletes the electrons in the 2DES underneath, allowing us to define the geometry of the device. The electron density in the surrounding regions, while not zero, is usually also reduced from its nominal value. When studying the QHE this can be undesirable because the filling factor depends on electron density. If the electron density within the device is less than everywhere else in the Hall bar, the filling factor will also be different; this can lead to difficulties in data collection and analysis.

In order to alleviate this problem, we anneal the gates at 4 K before cooling to base temperature. Because there is a bath of liquid He inside the dilution refrigerator, maintaining the temperature of the sample at 4 K is particularly easy. To anneal, we apply a negative voltage V_G^{anneal} to the gates and wait for tens of hours (usually $\sim$ 40–60 hours) with these conditions held constant. The gate voltage applied at this stage is the most negative voltage we anticipate applying during the actual experiment. After waiting the desired length of time the dilution refrigerator is cooled to base temperature, while still holding the gate voltage constant. We believe that while

at 4 K the electrons in the Si donor layer are able to rearrange in response to the annealing gate voltage. The 2DES directly underneath the gates remains depleted, but the surrounding regions are screened by the mobile electrons and the electron density in these regions is not reduced by the gate voltage.

After cooling to base temperature the gate voltage V_G can be, for the most part, varied as normal. Changing the V_G to be too negative ($V_G < V_G^{\text{anneal}}$) or too positive ($V_G \gtrsim V_G^{\text{anneal}} + 1\,\text{V}$) has been observed to lead to an increase in noise and/or to cause hysteresis in V_G. Hence we limit the gate voltage to the range $V_G^{\text{anneal}} \leq V_G < V_G^{\text{anneal}} + 1\,\text{V}$. In this range the electron density in the device remains nearly equal to the electron density in the full Hall bar. Thus the filling factor is essentially equal throughout the entire sample. This allows us to validly compare r_D and r_{XY}, as both reflect the physics of the same quantum Hall state with different types of confinement—a QPC for r_D and a wide Hall bar for r_{XY}.

The results of the annealing procedure with a QPC device geometry are shown in Figure 2.5 for two different ranges of magnetic field. The device configuration is the same as the one discussed in Chapter 4, with gates A1 and G3 annealed at $V_G^{\text{anneal}} = -2.7\,\text{V}$ and the remaining gates grounded (see Figure 2.2 for device geometry). Before annealing, r_D differs significantly from r_{XY}. In particular, r_D exhibits a larger slope in B (QHE plateaus shifted to lower B), indicating that the electron density in the QPC is less than in the full Hall bar. After annealing, the slope of r_D decreases to nearly equal that of r_{XY} and the various features of the two curves, such as the locations of the QHE plateaus, are closely matched in magnetic field position. Annealing has increased the electron density in the QPC to be nearly the same as

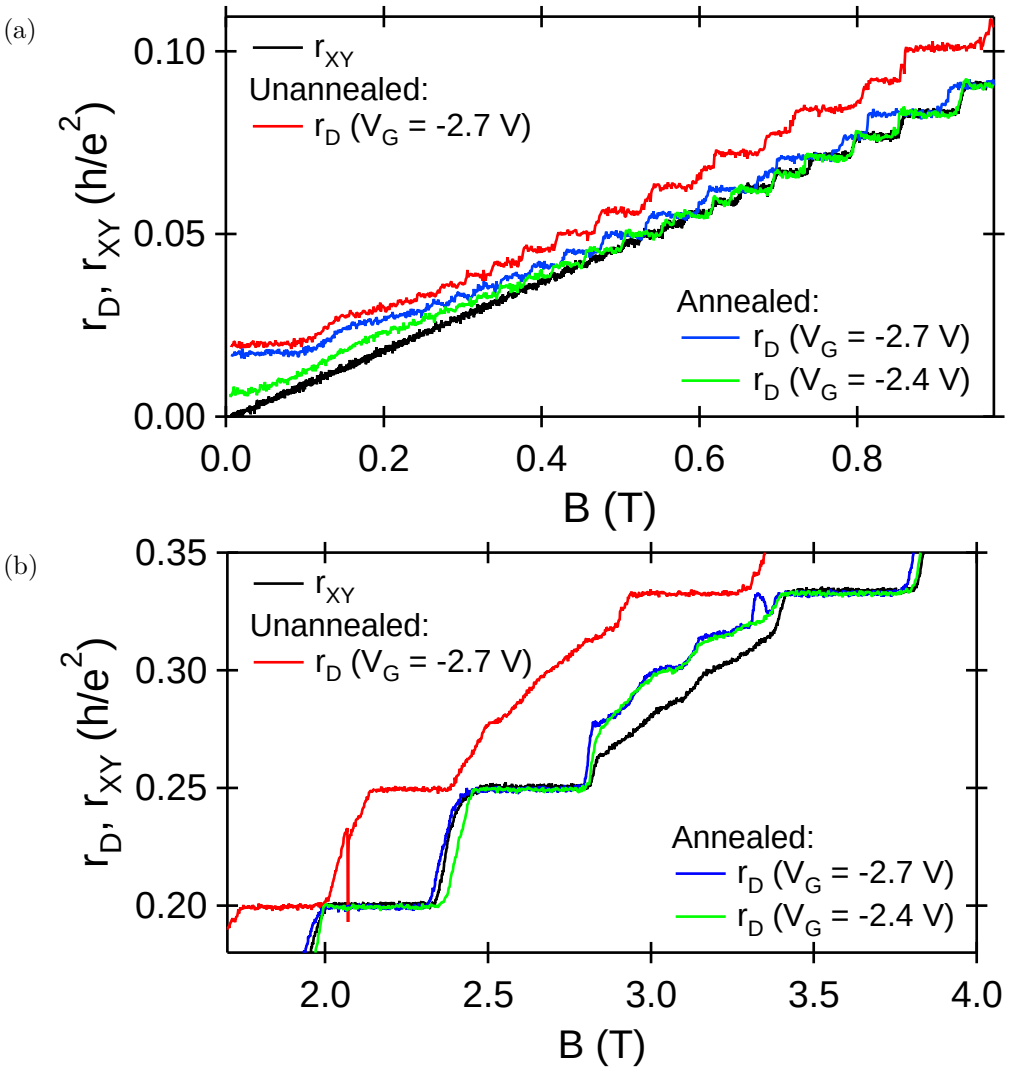


Figure 2.5: **Effect of annealing gates at low and high magnetic fields.** Effect of annealing gates at $V_G^{\text{anneal}} = -2.7\,\text{V}$ on r_D (colored curves), as compared to r_{XY} (black curve). (a) Comparison at low magnetic field. Increased resistance of r_D compared to r_{XY} at $B = 0$ is due to backscattering of electrons by the QPC. However, after annealing r_D converges to r_{XY} as B is increased. (b) Comparison at high magnetic field, in the presence of integer quantum Hall plateaus. For both regimes, r_D before annealing (red) differs from r_{XY} but closely matches after annealing (blue). Increasing the gate voltage to $-2.4\,\text{V}$ (green) improves the match to be nearly perfect.

the density throughout the rest of the Hall bar. Applying somewhat more positive gate voltage $V_G > V_G^{\text{anneal}}$ removes any remaining difference in the electron density. Measurements of the electron density in the Hall bar and in the QPC as a function of gate voltage and annealing condition are presented in Section 2.3. We note that the increased resistance of r_D compared to r_{XY} at $B = 0$ and between quantum Hall plateaus in Figure 2.5 is the result of backscattering caused by the QPC.

The success of the annealing technique is important to the experiments presented in Chapters 3 and 4. In Chapter 3 we study the tunneling conductance of quasiparticles tunneling between edge states in a QPC at $\nu = 5/2$. Because of the quasiparticle tunneling, r_D does not exhibit a quantized 5/2 plateau and, naively, one might be concerned that the 2DES in the QPC is not really at $\nu = 5/2$. However, we know from the success of the annealing procedure that the electron density is nearly uniform everywhere. Hence the fact that r_{XY} exhibits a quantized 5/2 plateau indicates that the filling factor is indeed $\nu = 5/2$ throughout the entire sample, including the QPC. In Chapter 4, breakdown of the QHE is studied as a function of QPC gate voltage (among other parameters). In this case it is important to be certain that the observed dependence is a result of the changing width of the QPC (calculated in Section 2.3) rather than changing electron density. Again, it is use of the annealing procedure that allows us to accurately perform this measurement.

Unfortunately, annealing has proven not to work in all GaAs heterostructures. We have found annealing to be effective in two similar heterostructures, both with double-sided doping and the Si set into GaAs quantum wells. In a third heterstructure, with single-sided doping and the Si in AlGaAs, annealing is not effective at maintaining

a uniform electron density. We suspect that the electrons at the Si donor sites are too tightly bound in this third sample to effectively move and screen the gate voltage at 4 K. For Al concentrations $x > 0.22$ the Si atoms form DX centers, which have very high electron binding energy of up to $\sim$ 100 meV [54]. However, the third heterostructure, in which annealing was not effective, has only $x = 0.104$. For $x < 0.2$, the Si donors instead form hydrogenic states with binding energies of up to 6 meV [54, 55]. This is a similar scale to the 3–5 meV binding energy of Si donors in GaAs [55, 56]. The electron effective mass increases in AlGaAs, but only slightly, going as $m^* = (0.067 + 0.083x)m_e$. All told there seems to be little reason to expect that the Si donors in AlGaAs with $x < 0.2$ behave significantly differently from those in GaAs. However, the empirical result is clear that annealing was not effective in the third heterostructure. Further testing of the annealing technique in additional heterostructures of various types will likely lead to a firmer conclusion regarding in which types of heterostructures annealing is effective.

Finally, we would like to note that the annealing time of $\sim$ 40–60 hours was chosen rather conservatively as a result of experimental inertia—we had always annealed for approximately that long and would rather wait a bit longer at 4 K than cool down, find out that the annealing wasn't quite effective, and have to do it all over again. However, preliminary evidence indicates that the annealing actually takes effect on the time scale of a few hours. If this is accurate, annealing the sample overnight is likely to be equally effective.

2.3 QPC Width and Electron Density

A useful experimental technique allows us to estimate the width of a QPC and also measure the electron density within the QPC. At zero magnetic field the QPC confinement potential causes the electrons to organize into quantum electrostatic subbands [46]. The number of these subbands passing through the QPC determines the resistance of the QPC (as each subband has conductance e^2/h per spin degree of freedom). This is illustrated by the steps in resistance around $-3.5\,\mathrm{V}$ in Figure 2.3. As the gate voltage is made more negative the number of subbands passing through the QPC is decreased, leading to a series of steps with (almost) quantized resistance.

As the magnetic field is increased, the nature of the subbands is affected by both the QPC electrostatic potential and the magnetic field. When the energy scales of the two are comparable, magnetoelectric subbands are formed [46]. These can be depopulated either by increasing the electrostatic confinement or by increasing the magnetic field. Going to even higher magnetic fields leads to the formation of Landau levels (magnetic subbands), at which point the effects of the electrostatic confinement are negligible.

For each of these three regimes the number of subbands passing through a QPC can be calculated for a given electrostatic potential. We will use the results for the simple cases of a parabolic potential and a square well potential [46]. In both cases, the QPC is modeled as an infinitely long channel with an infinitely deep potential. The integer number N_{QPC} of subbands in the QPC at or below the Fermi energy is,

for the parabolic potential,

$$N_{\mathrm{QPC}}^{\mathrm{par}} = \mathrm{Int}\left[\frac{1}{2} + \frac{k_F W}{4\sqrt{1+(W/2l_c)^2}}\right] \tag{2.1}$$

and, for the square well potential,

$$N_{\mathrm{QPC}}^{\mathrm{sq}} \approx \mathrm{Int}\left[\frac{\hbar k_F^2}{\pi e B}\left(\arcsin\left[\frac{W}{2l_c}\right] + \frac{W}{2l_c}\sqrt{1-\left(\frac{W}{2l_c}\right)^2}\right)\right] \qquad \text{for } 2l_c > W \tag{2.2a}$$

$$N_{\mathrm{QPC}}^{\mathrm{sq}} \approx \mathrm{Int}\left[\frac{1}{2} + \frac{\hbar k_F^2}{2eB}\right] \qquad \text{for } 2l_c < W. \tag{2.2b}$$

Here, $l_c = \hbar k_F/eB$ is the classical cyclotron radius at the Fermi energy. Hence, N_{QPC} is a function of the magnetic field. As B increases from zero the nature of the subbands changes from electrostatic to magnetoelectric to magnetic. The number of subbands also depends on the width W of the confinement potential and the Fermi wave vector k_F. The width of the parabolic potential $V(x) = m^*\omega_{\mathrm{par}}^2 x^2/2$ is defined as the width at the Fermi energy:

$$W_{\mathrm{par}} = \frac{2\hbar k_F}{m^*\omega_{\mathrm{par}}}. \tag{2.3}$$

We relate the Fermi vector to the electron density n_{QPC} in the QPC using the standard relation for a two-dimensional system:

$$k_F = \sqrt{2\pi n_{\mathrm{QPC}}}. \tag{2.4}$$

We do not directly measure the number of subbands in the QPC. However, we can measure the resistance r_D (and hence conductance) through the QPC. Each subband is effectively a one-dimensional state carrying current through the QPC. According to the Landauer-Büttiker formalism (discussed in Section 1.5) each of these has conductance $2e^2/h$. Here, the factor of 2 arises from the spin degeneracy, which

we accounted for differently in Section 1.5. The resistance of the QPC is thus given by

$$r_D = \frac{h}{2e^2}\frac{1}{N_{\mathrm{QPC}}}. \tag{2.5}$$

Although N_{QPC} technically only takes integer values, in order to fit Equation (2.5) to our data we allow N_{QPC} to be continuous. In this manner we can fit Equation (2.5) to r_D as a function of B and extract the width and electron density of the QPC. An example of this fit (for both confinement potentials) is shown in Figure 2.6. The kink in the square well fit at $\sim 0.2\,\mathrm{T}$ indicates the point at which the subbands switch from being dominated by the electrostatic confinement to be dominated by the magnetic field. This point is given by the condition $2l_c \approx W$.

It is instructive to consider the limits of Equations (2.1) and (2.2) at $B = 0$ and at high B. Simplifying at $B = 0$, we find for the parabolic potential

$$\begin{aligned}\lim_{B\to 0} N_{\mathrm{QPC}}^{\mathrm{par}} &= \mathrm{Int}\left[\frac{1}{2} + \frac{k_F W}{4}\right] \\ &\approx \frac{k_F W}{4},\end{aligned} \tag{2.6}$$

and for the square well potential

$$\begin{aligned}\lim_{B\to 0} N_{\mathrm{QPC}}^{\mathrm{sq}} &\approx \lim_{B\to 0} \mathrm{Int}\left[\frac{\hbar k_F^2}{\pi e}\frac{1}{B}\left(\arcsin\left[\frac{WeB}{2\hbar k_F}\right] + \frac{WeB}{2\hbar k_F}\right)\right] \\ &\approx \mathrm{Int}\left[\frac{\hbar k_F^2}{\pi e}\left(\frac{We}{2\hbar k_F} + \frac{We}{2\hbar k_F}\right)\right] \\ &\approx \frac{k_F W}{\pi}.\end{aligned} \tag{2.7}$$

If the electron density (and hence Fermi wave vector) is known, the width of the QPC can be simply calculated from the QPC resistance at zero magnetic field. The width estimate for the parabolic potential will be larger than that of the square well

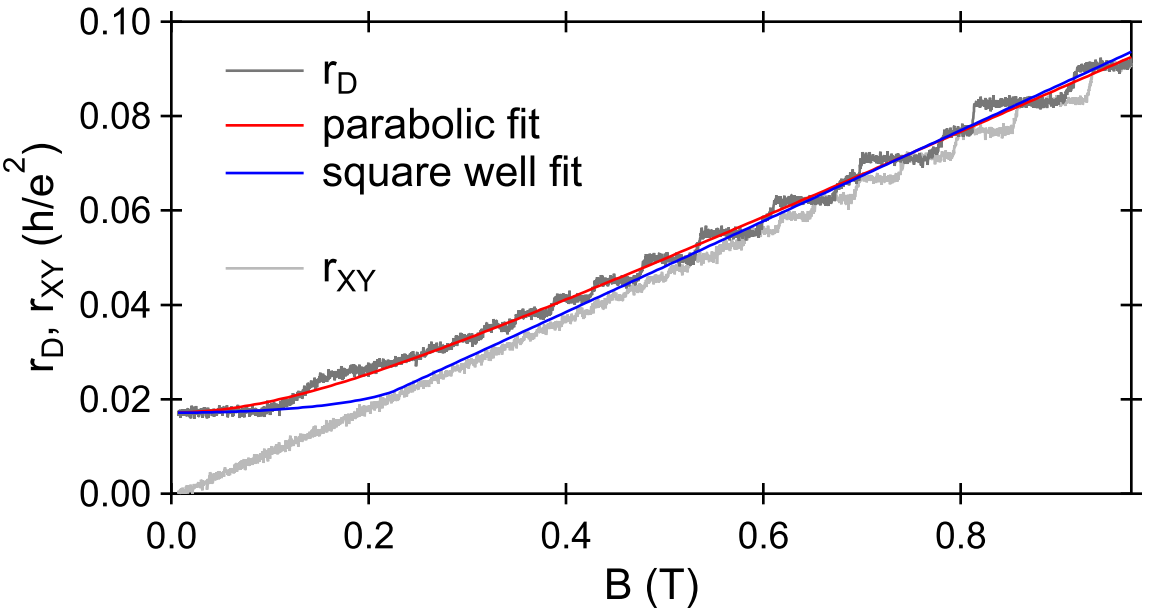


Figure 2.6: **Differential resistances and fits at low magnetic field.** Fits to r_D are performed using Equations (2.1), (2.2), and (2.5). The QPC is formed with gates A1 and G3 (see Figure 2.2) annealed at $-2.7\,\mathrm{V}$.

potential by a factor of $4/\pi$ because of the slightly different mathematics of the two potentials. The magnetic field dependence in the limit of large B is found to be

$$\begin{aligned}\lim_{2l_c \ll W} N_{\mathrm{QPC}}^{\mathrm{par}} &= \mathrm{Int}\left[\frac{1}{2} + \frac{k_F W}{4W/2l_c}\right] \\ &\approx \frac{\hbar k_F^2}{2eB}\end{aligned} \tag{2.8}$$

for the parabolic potential. For the square well potential the limiting form is already given by Equation (2.2b), which we simplify to

$$\begin{aligned}\lim_{2l_c \ll W} N_{\mathrm{QPC}}^{\mathrm{sq}} &\approx \mathrm{Int}\left[\frac{1}{2} + \frac{\hbar k_F^2}{2eB}\right] \\ &\approx \frac{\hbar k_F^2}{2eB}.\end{aligned} \tag{2.9}$$

The limiting form at high magnetic field is the same for both confining potentials and is independent of the width of the potential. Applying this result to Equation (2.5), we find

$$r_D = \frac{B}{e n_{\mathrm{QPC}}}, \tag{2.10}$$

which is the analog to the classical Hall effect, Equation 1.1. One can extract the electron density in the QPC from the high magnetic field slope of r_D. In addition, this confirms the claim above, that at high magnetic fields the electrostatic confining potential becomes irrelevant and the subbands become purely magnetic in nature. Because we have taken N_{QPC} to be continuous in this approximation we recover the result of the classical Hall effect. We know however, from Section 1.3, that the existence of Landau levels leads to the QHE. This analysis has neglected effects such as tunneling between edge states, which is discussed in Section 1.5 and Chapter 3.

Figures 2.7 and 2.8 show the QPC width and electron density resulting from fits of Equation (2.5) for an unannealed QPC and the same QPC annealed at $-2.7\,\mathrm{V}$,

respectively. In both cases the width of the QPC increases with more positive gate voltage. After annealing, this dependence is nearly linear over the range of gate voltages measured. When the QPC is not annealed, the electron density in the QPC is roughly 15% less than the density throughout the Hall bar, as measured by the slope of r_{XY} versus B. Annealing the QPC removes this discrepancy; after annealing the electron density in the QPC is within 1% of the density of the Hall bar. As discussed in Section 2.2, maintaining a uniform electron density, and hence a uniform filling factor, is an important requirement for the experiments presented in Chapters 3 and 4.

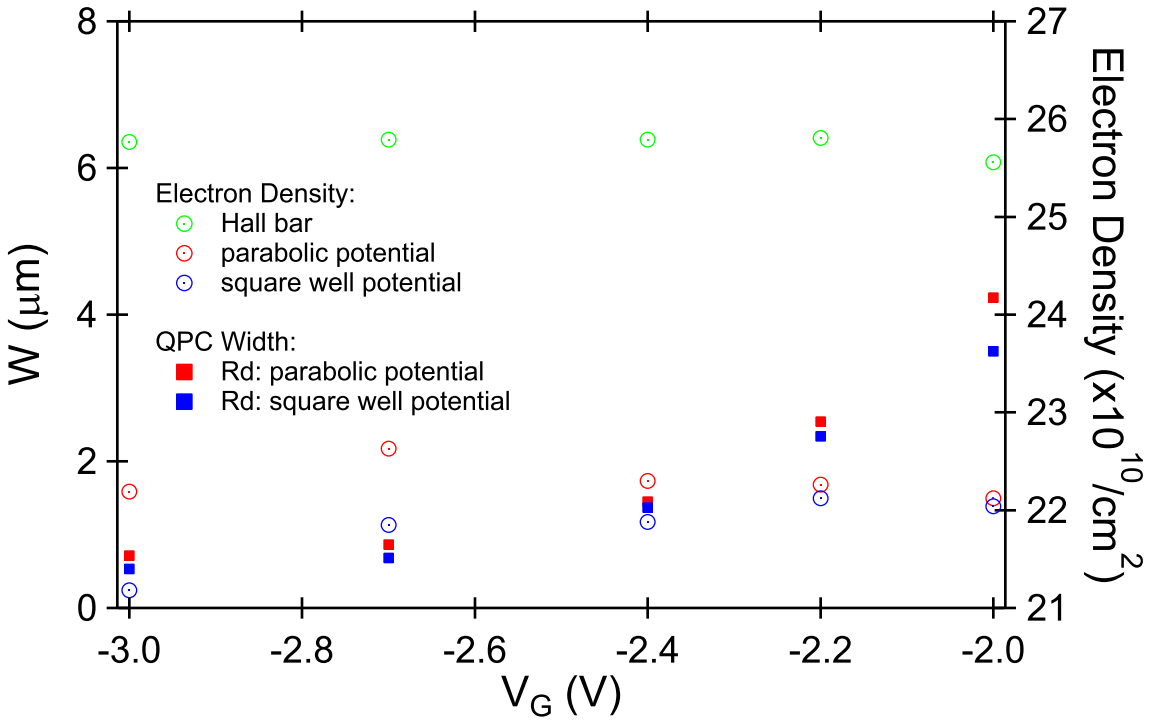


Figure 2.7: **QPC width and electron density extracted from fits to r_D with gates not annealed.** Values of QPC width W and electron density n_{QPC} in the QPC extracted with fits of Equation (2.5) at various gate voltages V_G. For comparison, electron density of the Hall bar, calculated from the slope of r_{XY} as a function of B is also presented. The QPC is formed with gates A1 and G3 (see Figure 2.2) with no annealing.

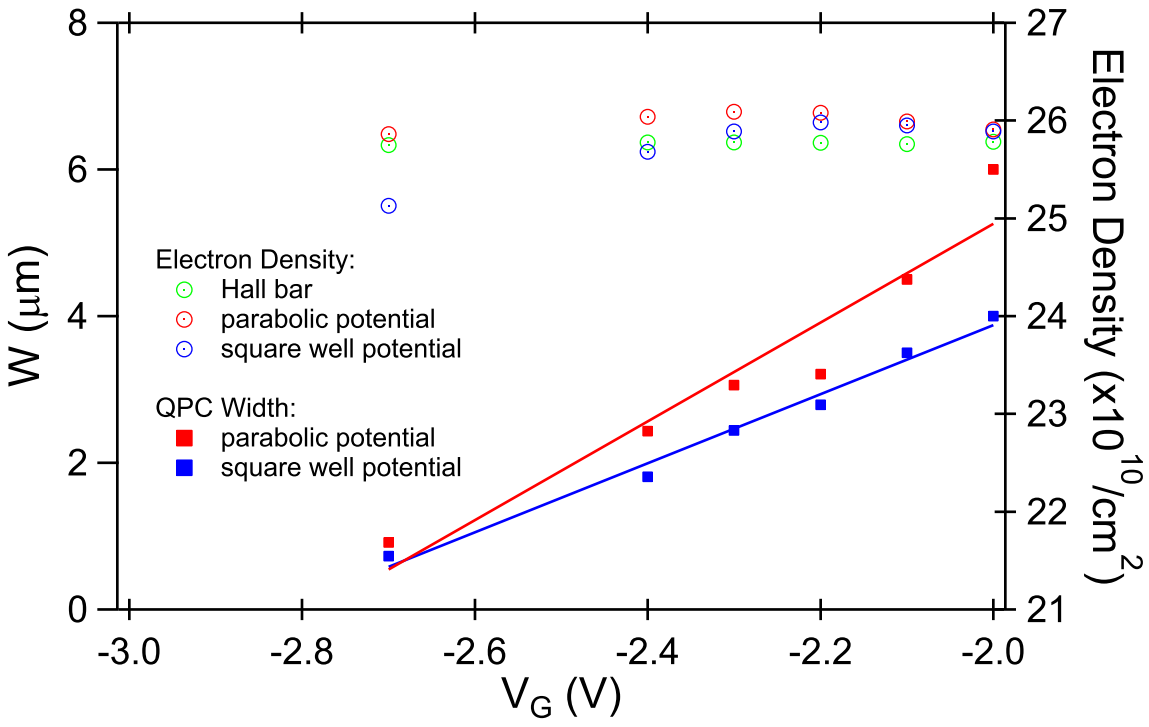


Figure 2.8: **QPC width and electron density extracted from fits to r_D with gates annealed.** Values of QPC width W and electron density n_{QPC} in the QPC extracted with fits of Equation (2.5) at various gate voltages V_G. For comparison, electron density of the Hall bar, calculated from the slope of r_{XY} as a function of B is also presented. The QPC is formed with gates A1 and G3 (see Figure 2.2) annealed at $-2.7\,\mathrm{V}$. The solid lines are linear fits of QPC width versus gate voltage.

Chapter 3

Quasiparticle Tunneling at $\nu = 5/2$

3.1 Controlling Tunneling Strength with Different Annealing Voltages

The collective interactions of a 2DES in a strong perpendicular magnetic field B, give rise to the fractional QHE [18], which is discussed in detail in Section 1.4. The states comprising the fractional QHE are characterized by fractional filling factors $\nu = n/(B/\Phi_0)$, where n is the electron density and $\Phi_0 = h/e$ is the quantum of magnetic flux. The $\nu = 5/2$ state [28] is of particular interest because it is one of only a few physically realizable systems thought to possibly exhibit non-abelian particle statistics [4, 36, 57–60]. Because of the energy gap in the bulk, motion of the quasiparticles that arise in the fractional QHE is generally constrained to one-dimensional chiral edge states [40]. However, if two edge states on opposite sides of the sample are brought close together, quasiparticles may tunnel between them. Studies of such

tunneling have led to measurements of the quasiparticle charge [20, 21] and creation of quasiparticle interferometers [6, 7]. The edge state picture is examined in Section 1.5. We demonstrate that by varying the gate annealing voltage, the tunneling strength of a long QPC can be tuned from strong to weak tunneling, while maintaining an electron density in the QPC equal to that elsewhere in the Hall bar.

Properties of the device measured and the GaAs/AlGaAs heterostructure are presented in detail in Section 2.1 and further explored throughout the rest of Chapter 2. We briefly summarize some of the characteristics here. The heterostructure has a measured mobility of $1 \times 10^7\,\mathrm{cm}^2/\mathrm{Vs}$ and electron density $n = 2.6 \times 10^{11}\,\mathrm{cm}^{-2}$. The mobility is only half that previously reported [50], but the cause of this degradation is unknown. The sample still exhibits a strong 5/2 fractional quantum Hall effect, with a quantized Hall plateau and vanishing longitudinal resistance, as shown in Figure 3.1. Metallic gates on the surface of the chip are biased negatively to deplete the underlying two-dimensional electron gas and induce tunneling between edge states. The gate pattern is shown in Figure 2.2; for the measurements presented in this section, gates G1, G2, G3, and G4 are energized, while the remaining gates (A1 and A2) remain grounded. This forms a long channel, similar to a QPC, with length $\sim 2.2\,\mu$m and width $\sim 1.2\,\mu$m.

A DC current I_{DC} of up to $\pm 10\,$nA with a $0.4\,$nA (RMS) AC modulation is applied to the source at one end of the Hall bar, with the drain at the other end. Using standard lock-in techniques at $17\,$Hz, measurements are made of the differential Hall resistance (r_{XY}) and differential longitudinal resistance (r_{XX}) at points remote from the QPC and of the differential diagonal resistance (r_D) across the QPC. These mea-

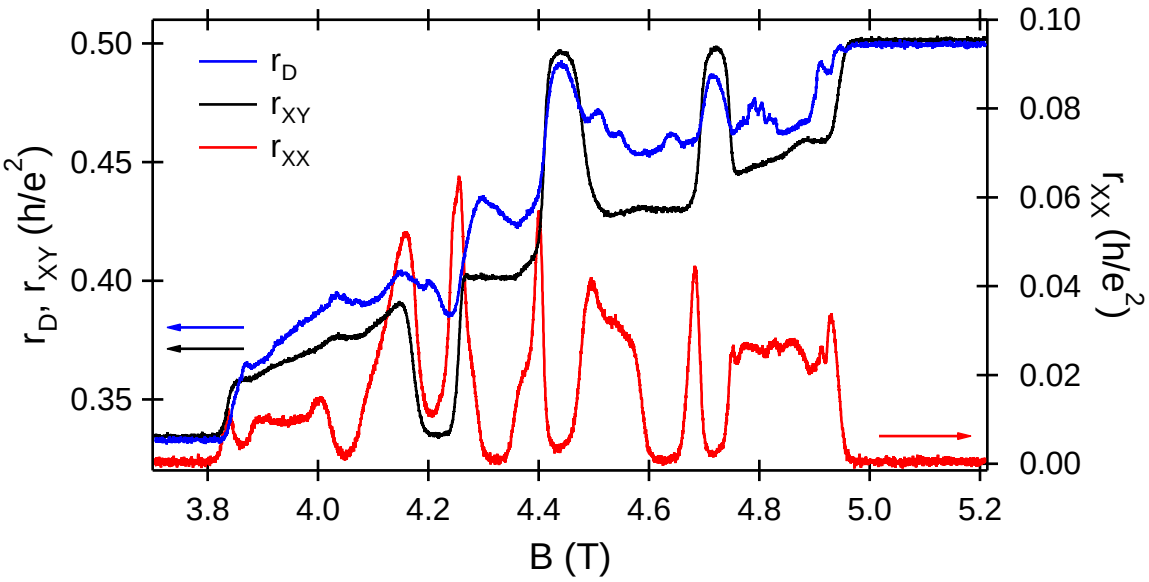


Figure 3.1: **Magnetic field dependence of r_{XX}, r_{XY}, and r_D between the $\nu = 3$ and 2 plateaus.** The differential Hall resistance r_{XY} (black), shows a well-quantized $\nu = 5/2$ plateau, centered at $B = 4.31\,\mathrm{T}$. The differential longitudinal resistance r_{XX} (red), measured separately, reaches zero for more than half of the plateau. Both r_{XY} and r_{XX} were measured in the Hall bar at points remote from the QPC. The differential diagonal resistance r_D (blue) exhibits well-quantized $\nu = 2$ and 3 integer quantum Hall plateaus. In the range of the 5/2 plateau r_D exhibits increased resistance compared to the quantized r_{XY} plateau, arising from tunneling of quasiparticles at the QPC.

surements are illustrated in Figure 2.4. As discussed in Section 2.1, r_D is very nearly the same as r_D^+, which was analyzed using the Landauer-Büttiker formalism in Section 1.5, and reflects the local physics of the QPC. Conversely, r_{XX} and r_{XY} are insensitive to the QPC and, instead, depend on the large-scale physics of the Hall bar far away from the QPC. Experiments are performed in a dilution refrigerator with a mixing chamber base temperature of $\sim 5\,\mathrm{mK}$. The temperatures T quoted in this chapter are electron temperatures, which track the mixing chamber temperature well down to $\sim 20\,\mathrm{mK}$. Lower electron temperatures are calibrated against thermally broadened Coulomb blockade peaks in a quantum dot and against quantum Hall features showing linear temperature dependence [53].

In order to preserve a near constant electron density throughout the entire sample, including the region of the QPC, we anneal the gates at 4 K for approximately 60 hours before cooling to base temperature. This procedure is discussed in detail in Section 2.2 and provides the advantage that the filling factor remains essentially uniform throughout the sample. As shown in Figure 3.1, the sample exhibits a strong $\nu = 5/2$ state (well-quantized r_{XY} with vanishing r_{XX}). Because the electron density, and hence filling factor, is the same in the QPC as in the full Hall bar we can be confident that the 5/2 state is also present in the QPC. This is consistent with our interpretation that the increased resistance in r_D compared to r_{XY} at $\nu = 5/2$ is caused by tunneling of quasiparticles. Results of three separate cool downs are presented—for annealing voltages $V_G^{\mathrm{anneal}} = -3$, -2.7, and $-2.4\,\mathrm{V}$. For each cool down we follow the measurement technique described in Radu et al. [50] to find the value in gate voltage for which the quasiparticle tunneling signature in r_D persists to the high-

est temperature. This procedure results in measurement voltages $V_G^{\text{meas}} = -2.53$, -2.434, and $-2.148\,\text{V}$, corresponding respectively to the three annealing voltages listed above. Measurements of tunneling across the QPC are performed at the center of the $\nu = 5/2$ plateau in r_{XY}, at $B = 4.31\,\text{T}$.

For each of these sets of annealing voltages we show low magnetic field measurements of r_D and r_{XY} in Figure 3.2. Each graph corresponds to a separate cool down with r_D plotted for gate voltages V_G equal to the annealing voltage V_G^{anneal} and to the measurement voltage V_G^{meas}. Each r_D curve is fit to a form describing depletion of magnetoelectric subbands in an infinitely long potential well. This fit is discussed in detail in Section 2.3. In all cases a square well potential (Equation (2.2)) is assumed, except for $V_G = -2.148\,\text{V}$ (annealed at $-2.4\,\text{V}$), in which case the parabolic potential (Equation (2.1)) provides a better fit. The characteristic width W of the potential is extracted from the fit and summarized for each gate voltage in Table 3.1. With the exception of $V_G = -2.148\,\text{V}$, all the widths are somewhat less than the lithographic width of $\sim 1.2\,\mu\text{m}$. We suspect that at $V_G = -2.148\,\text{V}$ the 2DES under the smallest gates, shown in Figure 2.2 (right), may not be completely depleted. This would allow the outer edge states, such as that corresponding to $\nu = 1$, to pass under the gates and the fit would hence return a much larger QPC width. This also explains why the $V_G = -2.148\,\text{V}$ data is better fit by the parabolic potential than the square well potential. We do, however, still expect the 5/2 quasiparticles to continue tunneling across the QPC at this gate voltage because, as will be shown in Section 3.2, the DC bias and temperature dependence of the QPC resistance r_D is fit well by theoretical predictions for weak quasiparticle tunneling.

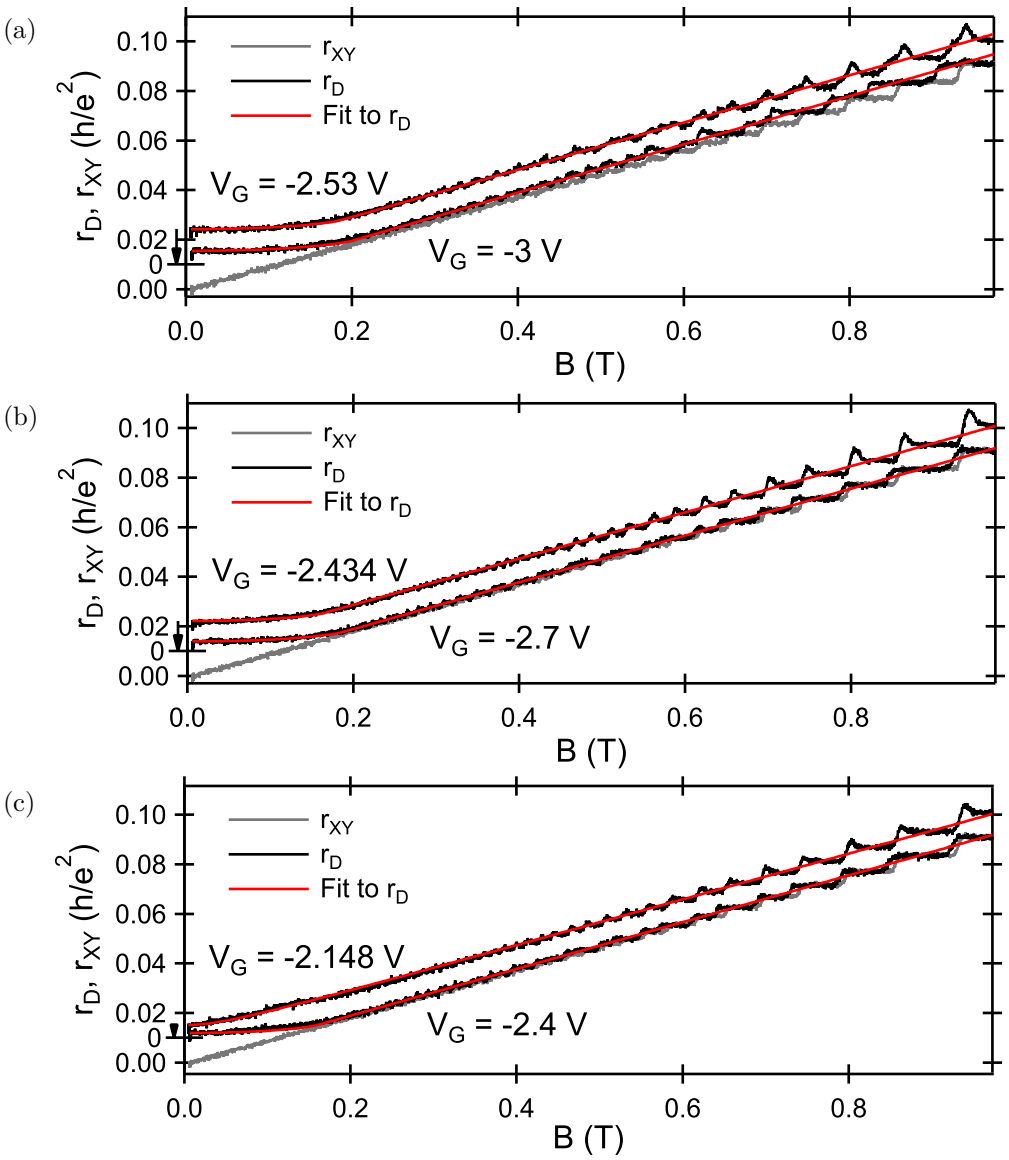


Figure 3.2: **Differential resistances and fits at low magnetic field and various gate voltages.** Low magnetic field measurements of r_{XY} and r_D at both $V_G = V_G^{\text{anneal}}$ and $V_G = V_G^{\text{meas}}$ for the three cool downs performed. The r_D data at $V_G = V_G^{\text{meas}}$ has been shifted vertically $0.01\,h/e^2$ for clarity. Red lines are fits to r_D based on an infinite square or parabolic potential well model.

Table 3.1: Summary of QPC widths extracted by fits to data in Figure 3.2.

Annealing Voltage (V)	-3		-2.7		-2.4	
V_G (V)	-3	-2.53	-2.7	-2.434	-2.4	-2.148
W (μm)	0.8	0.9	0.9	1.0	1.0	3.1

Tunneling of quasiparticles between edge states in the QPC produces characteristic nonlinearities in r_D. The differential tunneling conductance is peaked when the bias between the two edge states is equal to zero and falls to zero as the bias goes to infinity. This dependence is predicted [1, 61] based on the chiral Luttinger liquid model of edge states (discussed in Section 1.5) and has been confirmed experimentally [50, 62, 63]. We derive in Section 1.5 an expression in terms of r_D for the differential tunneling conductance g_t of quasiparticles in the weak tunneling regime given by

$$g_t = \frac{r_D - r_{XY}}{r_{XY}^2}. \tag{3.1}$$

This is Equation (1.35) with r_D^+ replaced by r_D (they are equal when $r_{XX} = 0$). Equation (1.35) is derived under the assumption that $r_D \approx r_{XY}$. This condition is roughly met for the measurements presented in this section, for which r_D deviates from r_{XY} by only up to $\sim 10\%$. Hence even if Equation (3.1) is not exact, it still gives us a rough idea of how the quasiparticle tunneling conductance is related to r_D.

An analytical form of the differential tunneling conductance has been calculated in the weak tunneling limit by Xiao-Gang Wen [1]. We present the exact result in Section 3.2, but note here that it predicts that the zero bias peak height follows a power law temperature dependence and the full-width at half maximum (FWHM) of

the peak in DC bias is linear in temperature. We are unaware of an analytic form for the tunneling conductance of strong tunneling. However experimental [50, 62] and numerical [61] results indicate that the zero bias peak flattens at low temperatures, indicating the innermost edge channel has been completely reflected. Dips to the sides of the peak, if present, become deeper and more pronounced for strong tunneling. For quasiparticle tunneling at $\nu = 5/2$, we experimentally find that at high bias r_D saturates to a background r_∞ which is larger than $0.4\,\mathrm{h/e^2}$; this background seems to be independent of DC bias and temperature under the measurement conditions chosen, although it does vary with annealing voltage.

This device was previously measured in a strong tunneling regime [50] when annealed at $-3\,\mathrm{V}$. Qualitatively similar DC bias and temperature dependence results, shown in Figure 3.3a, were obtained in this study, also for $V_G^{\mathrm{anneal}} = -3\,\mathrm{V}$ and for a similar measurement voltage ($V_G^{\mathrm{meas}} = -2.53\,\mathrm{V}$ in this case, compared to $-2.4\,\mathrm{V}$ previously). In particular, at the lowest temperatures the peak broadens and saturates at a value consistent with complete reflection of the $\nu = 5/2$ edge state when taking the r_∞ background into consideration. This assumes that the edge state of $\nu = 7/3$, which is the next plateau observed in Figure 3.1, is completely transmitted through the QPC. The peaks are off center by about $0.2\,\mathrm{nA}$ due to hysteresis in the sweep direction; all data shown in this chapter is measured with increasing DC bias and the small offset is accounted for when calculating peak height and width. In an effort to reach a weak tunneling regime in the QPC, measurements were performed at $V_G^{\mathrm{meas}} = -2.434\,\mathrm{V}$ with $V_G^{\mathrm{anneal}} = -2.7\,\mathrm{V}$ (Figure 3.3b) and $V_G^{\mathrm{meas}} = -2.148\,\mathrm{V}$ with $V_G^{\mathrm{anneal}} = -2.4\,\mathrm{V}$ (Figure 3.3c). We find that annealing at more positive gate voltage

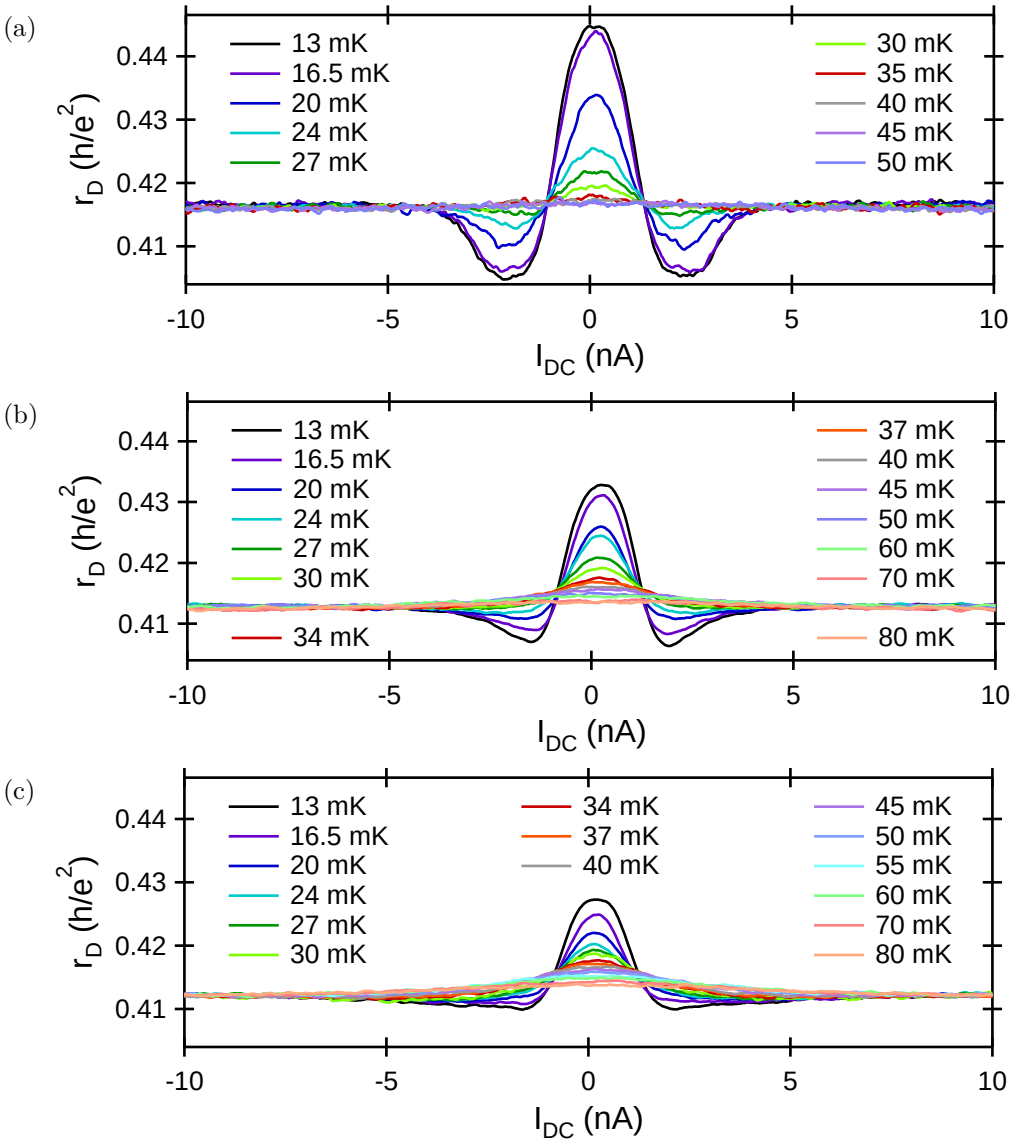


Figure 3.3: **DC bias and temperature dependence of r_D for three different annealing voltages.** (a) Measured at $V_G^{\text{meas}} = -2.53\,\text{V}$ with gates annealed at $V_G^{\text{anneal}} = -3.0\,\text{V}$. (b) Measured at $V_G^{\text{meas}} = -2.434\,\text{V}$ with gates annealed at $V_G^{\text{anneal}} = -2.7\,\text{V}$. (c) Measured at $V_G^{\text{meas}} = -2.148\,\text{V}$ with gates annealed at $V_G^{\text{anneal}} = -2.4\,\text{V}$. For all, each curve is measured with increasing DC bias.

leads to a reduction in the magnitude of the zero-bias tunneling peak. In addition, as we discuss below, a more positive gate voltage also produces curves that better exhibit the expected temperature scaling for weak tunneling. Hence, controlling the annealing voltage appears to be a promising method to tune the tunneling strength of a QPC.

For each set of measurements we check whether the data follows the peak height and FWHM scaling with temperature predicted for the weak tunneling regime. Peak height minus the temperature independent r_∞ background for each curve is shown in Figure 3.4a. Measurements for both $V_G^{\text{anneal}} = -3\,\text{V}$ and $V_G^{\text{anneal}} = -2.7\,\text{V}$ deviate from the power law temperature dependence, with the peak height saturating at lower temperatures. Conversely, the measurements with $V_G^{\text{anneal}} = -2.4\,\text{V}$ follow a power law dependence over nearly the entire temperature range, with an exponent that agrees with previous weak tunneling measurements at $\nu = 5/2$ [50]. The FWHM is plotted against temperature in Figure 3.4b. Here the picture is less clear, as all three sets of data show an upturn in FWHM at the lowest temperatures. However, at higher temperatures the FWHM after annealing at $V_G^{\text{anneal}} = -2.4\,\text{V}$ roughly follows a linear temperature scaling. Overall, more positive annealing voltages better produce the scaling and features expected from weak tunneling. We confirm the $V_G^{\text{anneal}} = -2.4\,\text{V}$ data does reflect weak tunneling by fitting to the full functional form of the differential tunneling conductance g_t in the weak tunneling regime. These results are presented and discussed in Section 3.2.

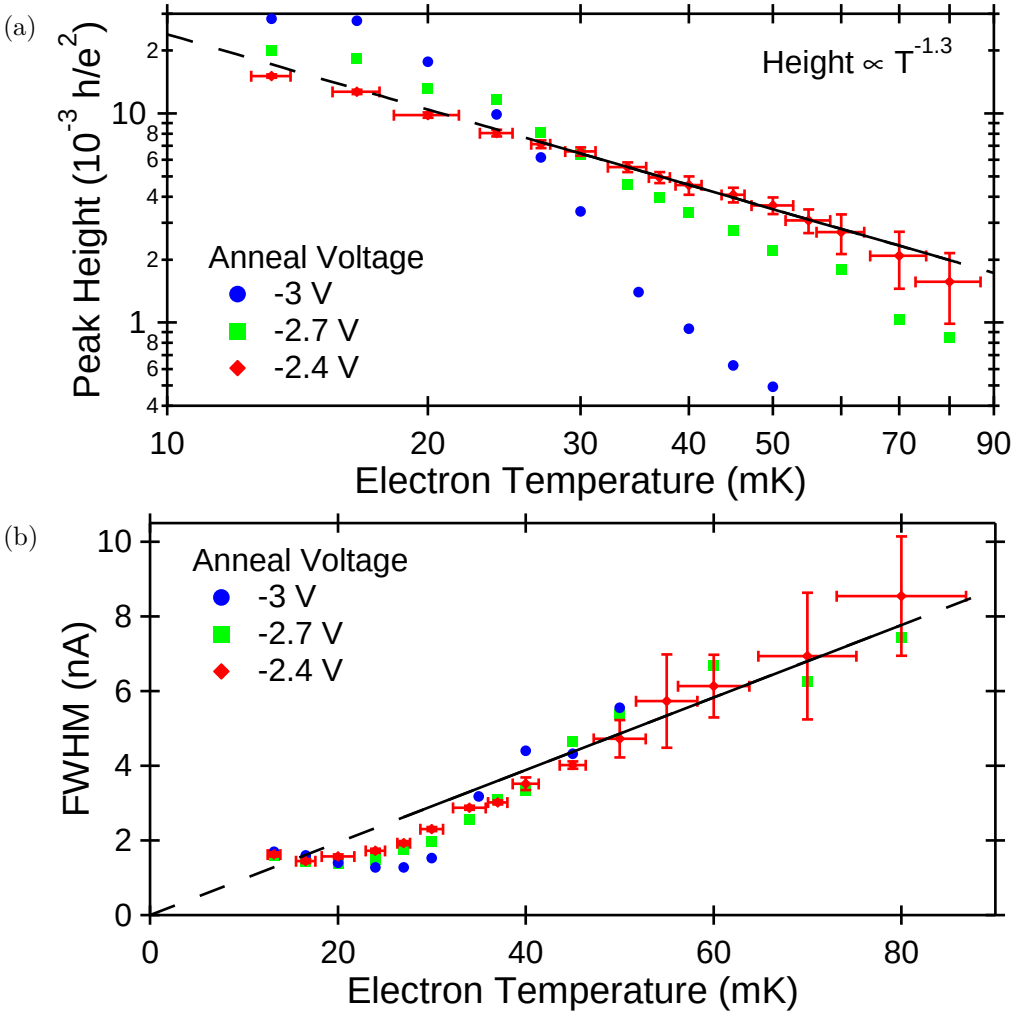


Figure 3.4: **Temperature scaling of data from Figure 3.3.** (a) Peak height at zero DC bias minus the r_∞ background (constant for each annealing voltage) as a function of electron temperature. The solid line is a power law fit in the range 27–80 mK to the peak height for $V_G^{\text{anneal}} = -2.4\,\text{V}$ and the dashed line extends the resulting power law to the entire range. (b) FWHM of each curve in Figure 3.3 as a function of electron temperature. The solid line is a linear fit in the range 27–80 mK to the FWHM for $V_G^{\text{anneal}} - 2.4\,\text{V}$ and the dashed line extends the resulting line to the entire range. For both, error bars are shown only for the measurements performed with annealing voltage $V_G^{\text{anneal}} = -2.4\,\text{V}$.

3.2 Tunneling Conductance in the Weak Tunneling Regime

In this section, we study the $\nu = 5/2$ state in two different QPC geometries, and present temperature and DC bias dependence of quasiparticle tunneling conductance across each QPC in the weak tunneling regime. By fitting these results to the theoretical form predicted by chiral Luttinger liquid theory [1], we extract the quasiparticle charge e^* and interaction parameter g. The resulting e^* is in agreement with the predicted value of $e/4$, and the value of g best agrees with that predicted for the abelian 331 wave function. Fixing g at the values predicted by other proposed wave functions produces fits that are qualitatively and quantitatively worse. In addition, qualitative features of the DC bias dependence also favor the 331 wave function.

A number of different ground state wave functions have been proposed for the 5/2 state, some with non-abelian statistics and some with prosaic abelian statistics (refer to Section 1.4 for a discussion of particle statistics). Were the existence of novel non-abelian statistics confirmed, it would be an exciting discovery of a new state of matter and would possibly enable topological quantum computation [34]. A great deal of theoretical and experimental work has recently focused on the 5/2 state [5, 10–12, 50, 64–77]. Experimentally, the quasiparticle charge e^* has been found to be consistent with the predicted value e/4 [5, 50, 75]. Numerical simulations indicate a preference for the non-abelian Pfaffian and anti-Pfaffian wave functions over various abelian wave functions [67, 70, 72, 76, 78]. The degree of electron spin polarization also provides valuable information about the wave function, but experimental results

are contradictory [73, 74, 77]. Recent experimental results from an interferometer have been interpreted as evidence for non-abelian statistics [10, 11]. The observation of a counter-propagating neutral mode is also most easily explained by the existence of a non-abelian state [12].

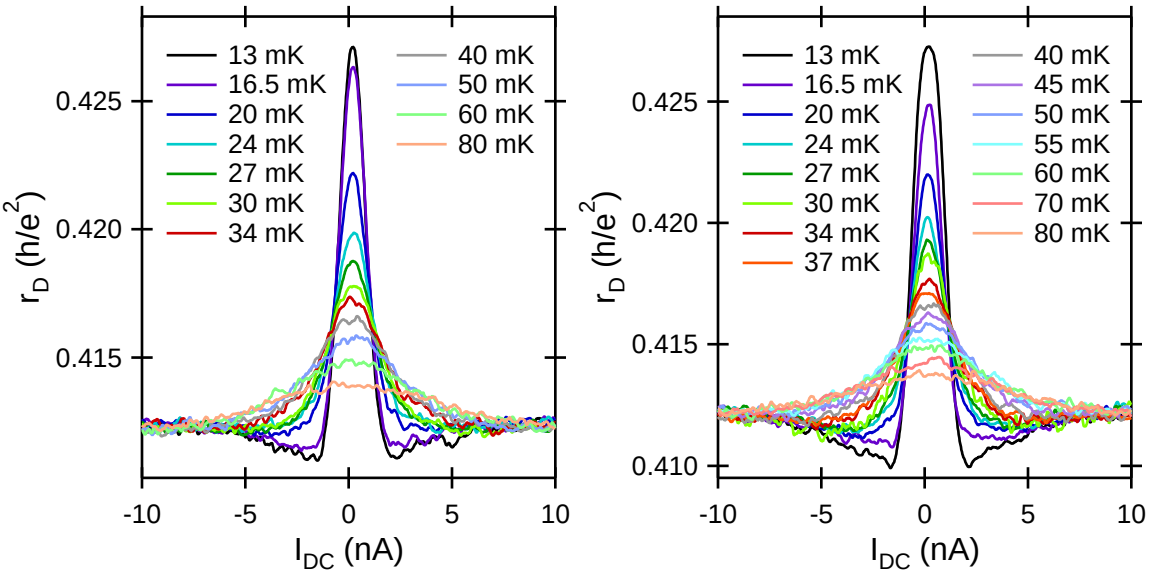
We measure the device explored in Section 3.1, with two different QPC geometries both in the weak tunneling regime. Geometry A is a short QPC of nominal width $\sim 0.6\,\mu$m, formed by energizing gates A1, G3, and G4 with the remaining gates grounded. Geometry B is a long channel of nominal width $\sim 1.2\,\mu$m and length $\sim 2.2\,\mu$m, formed by energizing gates G1, G2, G3, and G4, with the remaining gates grounded. This is the same geometry as is discussed in Section 3.1. See Figure 2.2 for details of the device geometry and gate labels. For both geometries we use V_G to refer to the gate voltage on the energized gates, which differ between the two geometries. Although Geometry B is not a traditional QPC (the constriction is extended instead of being limited to a 'point'), it still exhibits weak quasiparticle tunneling; for convenience we refer to both geometries as QPCs. The measurement setup is illustrated in Figure 2.4 and discussed in Section 3.1.

We follow the annealing procedure described in Section 2.2 in order to preserve the same electron density, and hence filling factor, both inside and outside the QPC. This is important in order to be confident that the measurements reflect tunneling of $\nu = 5/2$ quasiparticles from one chiral edge state to the other. Each geometry is annealed separately (each during a different cool-down) for approximately 60 hours before cooling to base temperature. For each geometry the gates listed above are annealed at $V_G^{\text{anneal}} = -2.4\,\text{V}$, while leaving the other gates grounded. After annealing

and lowering the temperature, the gate voltage is constrained to the range $-2.4\,\mathrm{V} \leq V_G \leq -1.8\,\mathrm{V}$. This identical device in Geometry B has been previously measured in the strong tunneling regime [50] and we have repeated these measurements with similar results in Section 3.1. Here we anneal at a less negative gate voltage enabling us to access the weak tunneling regime to much lower temperature.

For each geometry we measure the DC bias and (electron) temperature dependence of r_D at $\nu = 5/2$, as shown in Figures 3.5a and 3.5b. The peak positions deviate from $I_{DC} = 0$ by $\sim 0.2\,\mathrm{nA}$ because of hysteresis in the sweep direction; all measurements shown in this chapter are taken with increasing DC bias, and the small offset is subtracted when fitting. The magnetic field and gate voltage are chosen, following the measurement technique described in Radu et al. [50], to maximize the temperature range exhibiting a zero-DC-bias peak and to minimize variations in the background resistance. In particular, the magnetic field is set to 4.31 T, which is the center of the r_{XY} plateau for $\nu = 5/2$. Figure 3.6 illustrates the procedure for choosing gate voltage for Geometry B. Figures 3.6a and 3.6a show r_D as a function of I_{DC} and V_G for base temperature and 30 mK respectively. The degree to which the peak height and high DC bias background change with temperature depends on the gate voltage. We choose V_G such that the high DC bias background remains constant in temperature. This gate voltage also leads to the peak persisting to the highest temperatures. The full DC bias and temperature dependence displayed in Figure 3.5 is measured at $V_G = -2.1\,\mathrm{V}$ for Geometry A and at $-2.148\,\mathrm{V}$ for Geometry B.

In the limit of weak quasiparticle tunneling, r_D is linearly related to the differential



(a) (b)

Figure 3.5: **DC bias and temperature dependence of r_D for two gate geometries.** (a) Differential diagonal resistance r_D measured in Geometry A with an applied gate voltage of $V_G = -2.1\,\mathrm{V}$ as a function of DC bias I_{DC} (x-axis) and electron temperature T (colored curves). (b) Differential diagonal resistance r_D measured in Geometry B with $V_G = -2.148\,\mathrm{V}$ as a function of DC bias I_{DC} (x-axis) and electron temperature T (colored curves). For both, the magnetic field is set to the center of the $\nu = 5/2$ plateau in r_{XY} (at $B = 4.31\,\mathrm{T}$) and the sample is annealed at $-2.4\,\mathrm{V}$ as discussed in the text.

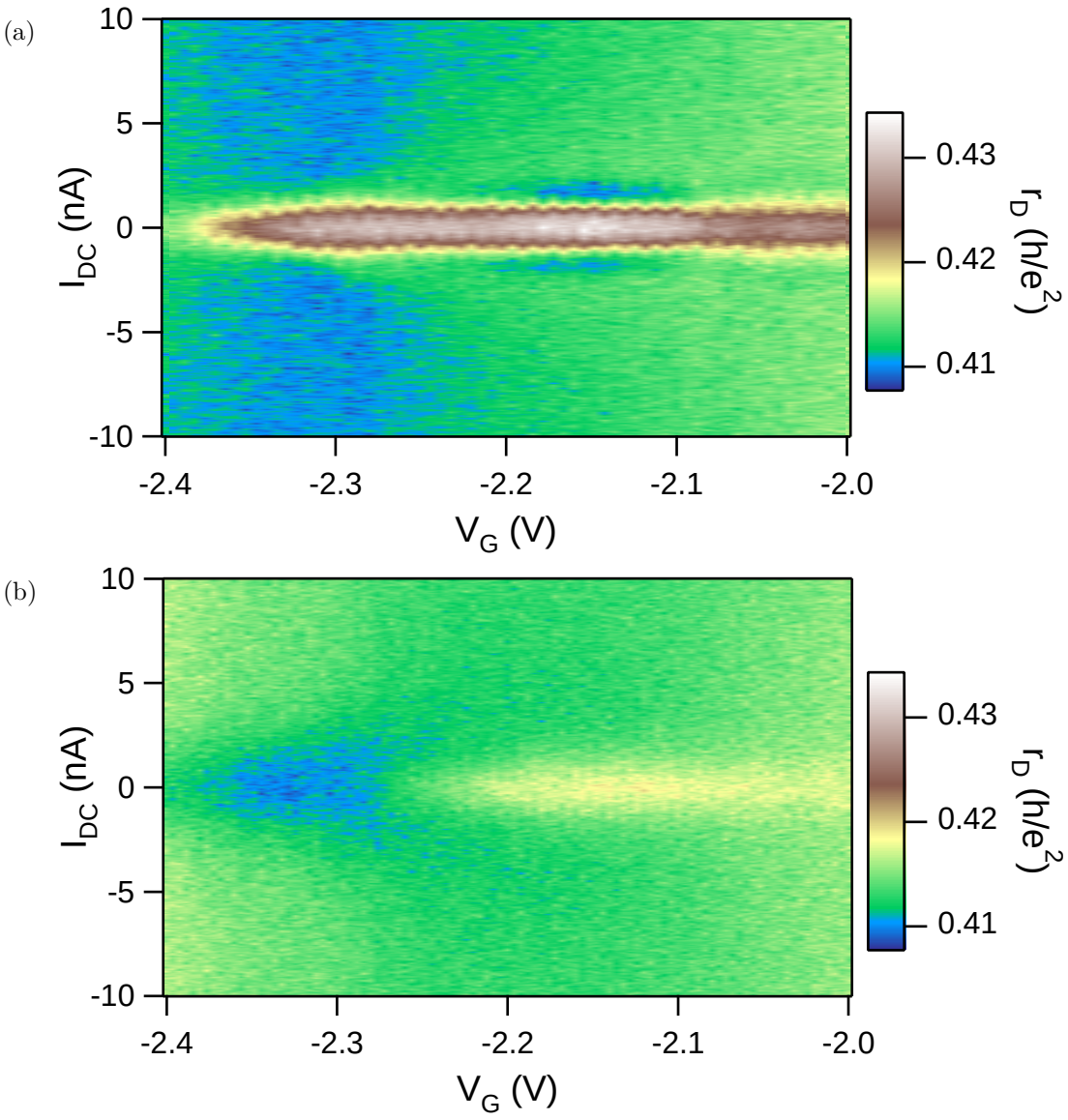


Figure 3.6: **Differential resistance r_D as a function of DC bias current and gate voltage for Geometry B.** (a) Measured at 13 mK electron temperature. (b) Measured at 30 mK electron temperature. The full temperature dependence of r_D is measured at $V_G = -2.1$ V for Geometry A and -2.148 V for Geometry B. At these gate voltages the DC bias peak persists to the highest temperature and the background r_∞ remains constant with temperature.

tunneling conductance g_t by

$$g_t = \frac{r_D - r_{XY}}{r_{XY}^2}, \tag{3.2}$$

provided that the sample is in an integer or fractional quantum Hall state. This result is derived in Section 1.5 as Equation 1.35. By converting r_D to g_t, we are able to fit using the theoretical form for weak quasiparticle tunneling in an arbitrary fractional quantum Hall state [1]:

$$g_t(I_{DC}, T) = AT^{2g-2} F\left(g, \frac{e^* I_{DC} r_{XY}}{k_B T}\right). \tag{3.3}$$

The quasiparticle charge e^* and interaction parameter g are physical constants characterizing the fractional quantum Hall state, while the fit parameter A accounts for the tunneling amplitude. The function F has a complicated functional form, given by

$$\begin{aligned} F(g,x) = {} & B\left(g + i\frac{x}{2\pi}, g - i\frac{x}{2\pi}\right) \\ & \times \left\{\pi \cosh\left(\frac{x}{2}\right) - 2\sinh\left(\frac{x}{2}\right) \mathrm{Im}\left[\Psi\left(g + i\frac{x}{2\pi}\right)\right]\right\}, \end{aligned} \tag{3.4}$$

where B is the Euler beta function and Ψ is the digamma function. The function F is peaked at zero bias and approaches zero at infinite DC bias. For $g < 1/2$, it exhibits minima on the sides of the zero-bias peak, while these minima are absent for $g \geq 1/2$. Equation (3.3) predicts that the zero-bias peak height follows a power-law temperature dependence and that the full-width at half maximum (FWHM) in DC bias is linear in temperature. Like Radu et al. [50] and as seen in Figure 3.5, we find a background resistance r_∞, which is larger than the expected quantized value $0.4\,\mathrm{h/e^2}$. However, this background is independent of DC bias and temperature

under the measurement conditions chosen and hence we also treat r_∞ as a single fit parameter.

The scaling behavior of r_D with temperature is illustrated in Figure 3.7. The results for both geometries follow the expected scaling over most of the temperature range measured. However, deviations are observed at the lowest temperatures. For both geometries the peak height begins to saturate when temperature is decreased below $\sim 20\,\mathrm{mK}$. Both geometries also exhibit a dip in the FWHM, with the actual FWHM falling below the value expected from a linear temperature dependence at temperatures less than $\sim$ 30 to 40 mK. These deviations are associated with the broadening and flattening of the peak associated with strong tunneling [50, 61, 62]. Fits to peak height over the range in temperature in which it follows a power law result in an exponent $(2g - 2)$ of -1.3 for Geometry A and -1.2 for Geometry B. Solving for g gives $g = 0.35$ for Geometry A and $g = 0.4$ for Geometry B. These values are within error bars of those determined more precisely with a full fit to Equation (3.3), as discussed below.

A least-squares fit of Equation (3.3) to the r_D measurements for both geometries is shown in Figure 3.8. We fit over multiple temperatures simultaneously, but limit the temperature range for fitting to $20 - 80\,\mathrm{mK}$ for Geometry A and to $27 - 80\,\mathrm{mK}$ for Geometry B to ensure that the strong tunneling regime is excluded. However, we note that the fitting results do not change significantly if different temperature ranges are chosen. Within the chosen range, measurements for all temperatures are fit *simultaneously* with the same fitting parameters. The best fit for Geometry A returns $e^* = 0.25e$ and $g = 0.42$ and for Geometry B returns $e^* = 0.22e$ and $g = 0.34$.

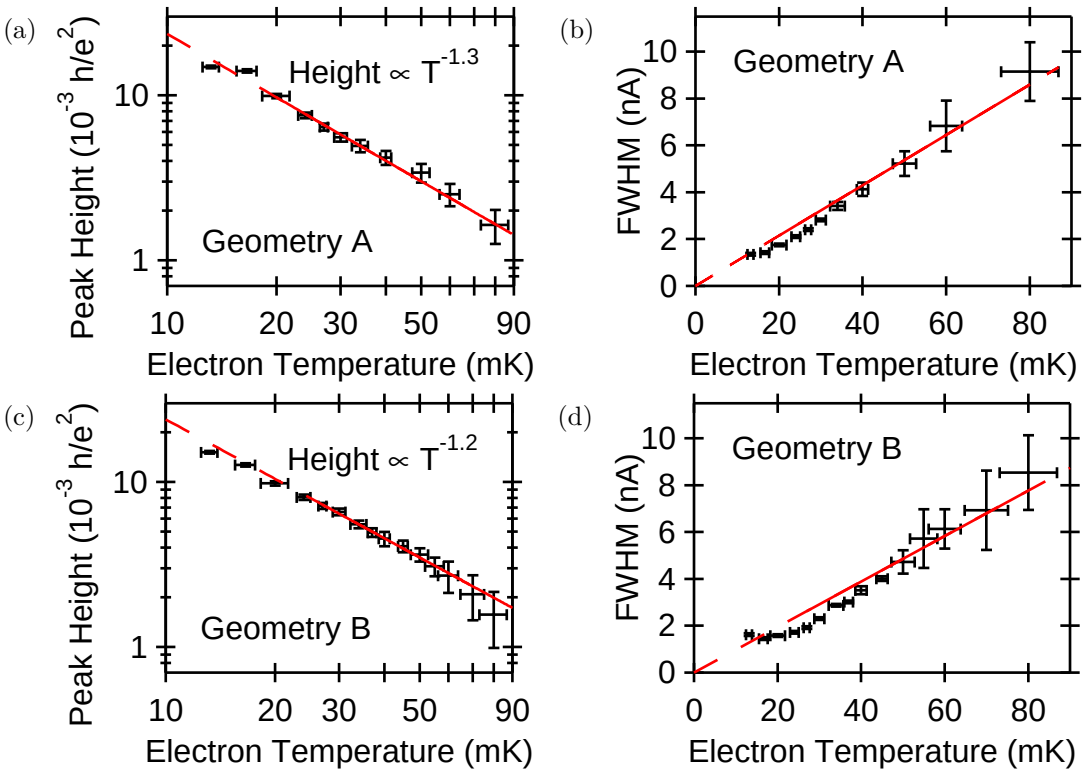


Figure 3.7: **Scaling of peak height and full width at half-maximum with temperature.** (a) Peak height at zero DC bias minus the constant r_∞ background against electron temperature for Geometry A. The solid red curve is a power law fit in the range 20–80 mK and the dashed red curve extends the resulting power law to the entire range. Below 20 mK the peak height begins to saturate, indicating a transition from weak to strong tunneling. (b) FWHM of the zero DC bias peak against electron temperature for Geometry A. The solid red curve is a linear fit in the range 20–80 mK with the y-intercept held to zero, and the dashed red curve extends the resulting line to the entire range. Deviation from the linear scaling predicted by weak tunneling theory occurs at lower temperatures, again indicating a transition to strong tunneling. (c), (d) As in parts (a) and (b), respectively, for Geometry B. Fits are performed over the 27–80 mK temperature range. For both geometries, the power law of the peak height temperature scaling agrees with the best fit value of g within error bars.

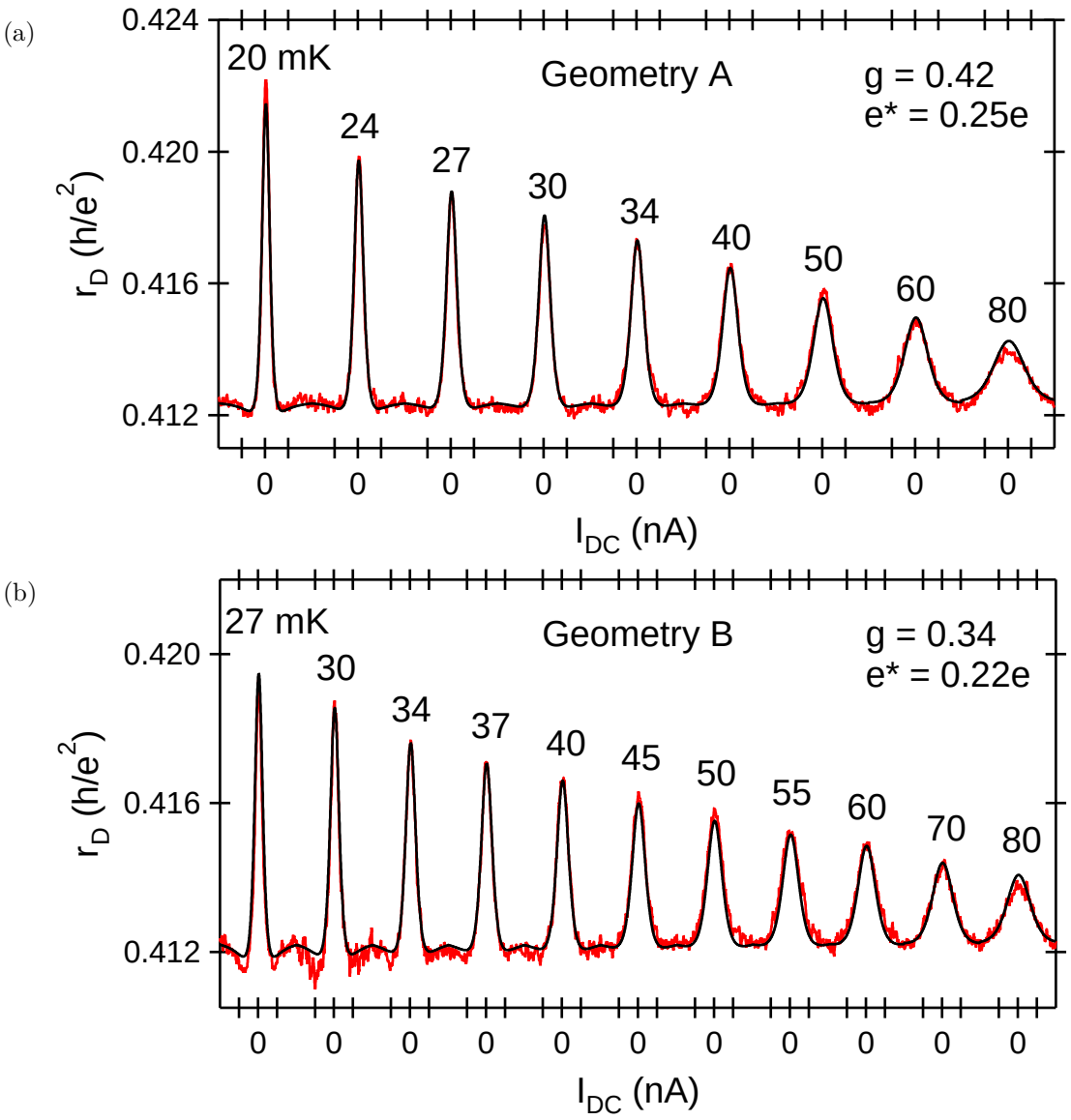


Figure 3.8: **Least-squares best fit of the theoretical form of weak tunneling conductance to r_D.** (a) Fit of Equation (2) to r_D measured in Geometry A, with all temperatures in the range 27–80 mK fit simultaneously. (a) Fit of Equation (2) to r_D measured in Geometry B, with all temperatures in the range 20–80 mK fit simultaneously. For both, tunneling conductance peaks at each electron temperature (labeled on graph) are concatenated to produce a single curve. Experimental results are red and the fit is black. Ticks on the horizontal axis indicate 0 and ±5 nA for each temperature.

These values are similar to those obtained in a previous weak tunneling measurement [50] ($e^* = 0.17e$, $g = 0.35$), but we find e^* much closer to the predicted value of $e/4$.

In order to understand the level of confidence we can place in these fit results, we fix (e^*, g) pairs over a range of values and fit to the weak tunneling form, allowing A and r_∞ to vary. The residual of each fit is divided by the measurement noise of $\sim 2 \times 10^{-4}\, h/e^2$ and plotted against the (e^*, g) pairs in Figure 3.9. Pairs of (e^*, g) values with a fit residue no larger than the noise fall within the "1" contour. Also indicated are the pairs of (e^*, g) values corresponding to the proposed wave functions listed below.

Alternatively, we can examine the reduced chi-squared of fits for a similar matrix of fixed (e^*, g). The results are shown in Figure 3.10. The reduced chi-squared is defined as chi-squared of a fit divided by the number of degrees of freedom in the fit, where the number of degrees of freedom is the number of data points (7200 for Geometry A and 8800 for Geometry B) minus the number of fit parameters. One might ideally expect to find a reduced chi-squared equal to 1, at least in the absence of systematic errors, corresponding to a p-value of $\sim 50\%$. Lower values of reduced chi-squared correspond to higher p-values. Proper statistical interpretation of reduced chi-squared and p-values is inherently complex, and in our case complicated by a number of factors: the fitting function is not linear, we do not directly measure or account for systematic errors, and the fitting function is a first order perturbation result (and hence is missing higher-order terms). Hence, while the reduced chi-squared results provide additional confidence that the fits within and near the "1" contour of Figure 3.10 are reasonable, we caution against drawing firm conclusions from these

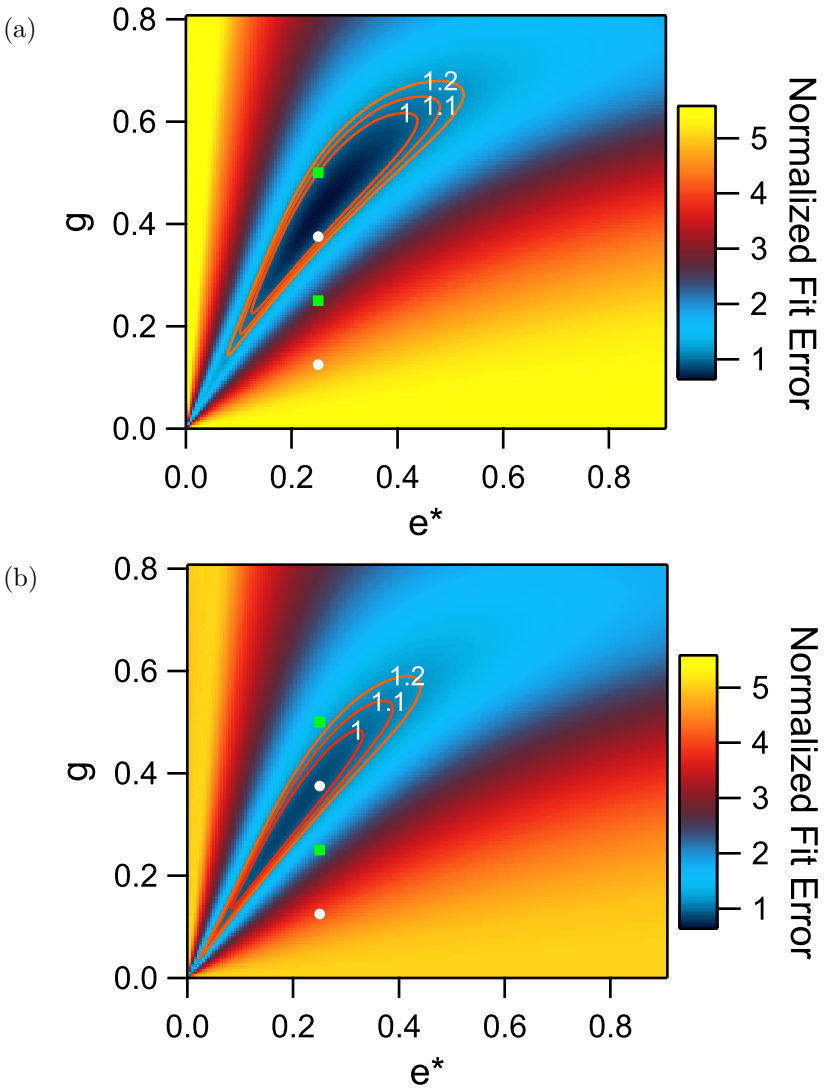


Figure 3.9: **Matrix of fit residual divided by the experimental noise for fixed pairs of (e^*, g).** (a) Results for Geometry A. (b) Results for Geometry B. For both, pairs of (e^*, g) corresponding to proposed non-abelian states (green squares) and abelian states (white circles) are plotted; see the text for more details. Contours of the normalized fit error are included as guides to the eye.

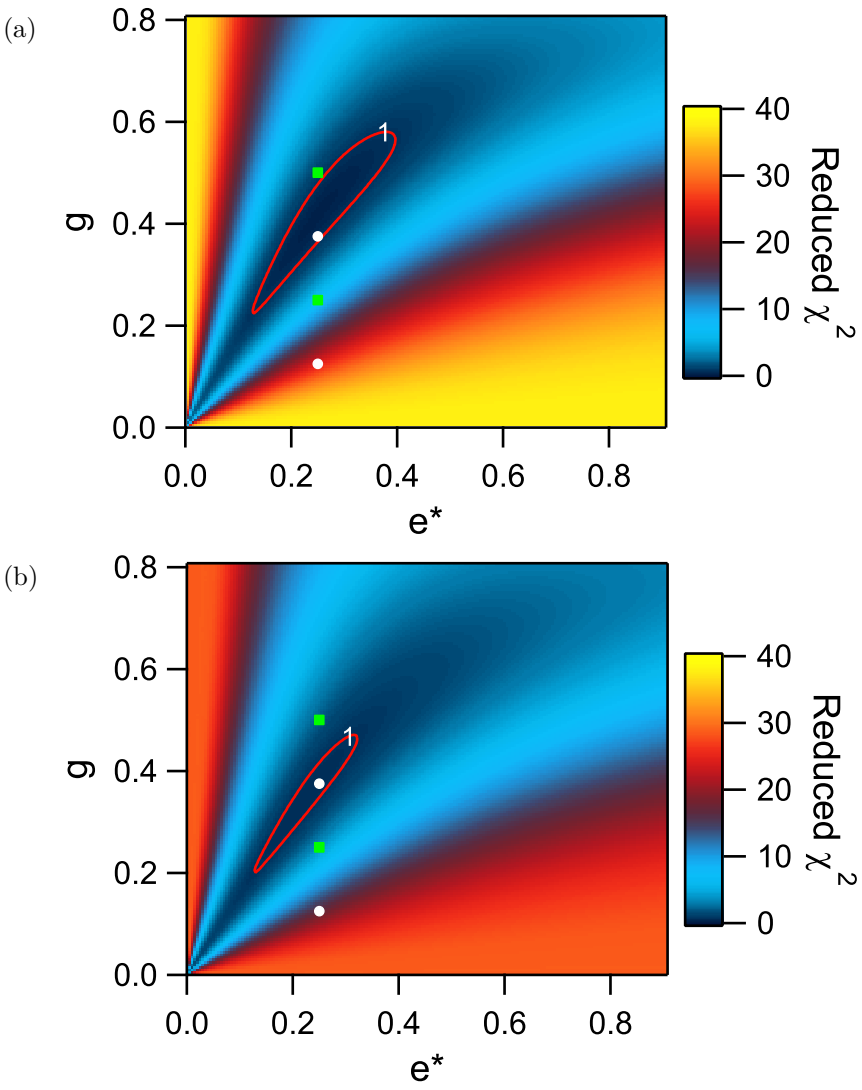


Figure 3.10: **Matrix of reduced chi-squared of fits for fixed pairs of** (e^*, g)**.** (a) Results for Geometry A. (b) Results for Geometry B. For both, pairs of (e^*, g) corresponding to proposed non-abelian states (green squares) and abelian states (white circles) are plotted; see the text for more details. A contour at reduced chi-squared equal to 1 is included.

results alone.

The simplest interpretation of our data is that the value of g derived from the fits directly reflects the nature of the quasiparticles, and therefore the wave function, of the 5/2 state in our system. This allows us to distinguish between competing proposals by their expected value of g. Proposed abelian states include the 331 [79, 80] with $g = 3/8$, and K = 8 [81] with $g = 1/8$. Non-abelian states include the Pfaffian [57] with $g = 1/4$, the particle-hole conjugate anti-Pfaffian [59, 60] with $g = 1/2$, and $\mathrm{U}(1) \times \mathrm{SU}_2(2)$ [58] with $g = 1/2$. All these states support quasiparticles both of charge $e/4$ and charge $e/2$. However, tunneling measurements are expected to be dominated by quasiparticles of charge $e/4$ [82]. Hence we examine the fits of Equation (3.3) with e^* fixed at $e/4$ and g fixed at a value predicted by one of the proposed states: 1/8, 1/4, 3/8, or 1/2. Of these four options, fits with $g = 3/8$ and $g = 1/2$ produce the lowest residuals, as can be seen in Figure 3.9; fits for $g = 1/8$ and 1/4 are very poor, both quantitatively and qualitatively.

Fits for the two geometries with e^* and g fixed to the values predicted by the various proposed wave functions are shown in Figures 3.11 and 3.12; also included are the best fits with e^* and g allowed to vary. Of the fits with fixed values, those with $g = 3/8$, corresponding to the abelian 331 state, are the best. These fits follow the data closely, including the temperature dependence of the peak height and the obvious minima on either side of the main peak. For $g = 1/2$, the minima on either side of the peak are absent from the fits, and peak heights are not well described. Fits with g fixed to 1/4 or 1/8 are clearly much worse. Our ability to better discriminate between $g = 1/2$ and $g = 3/8$ than with previous weak tunneling measurements

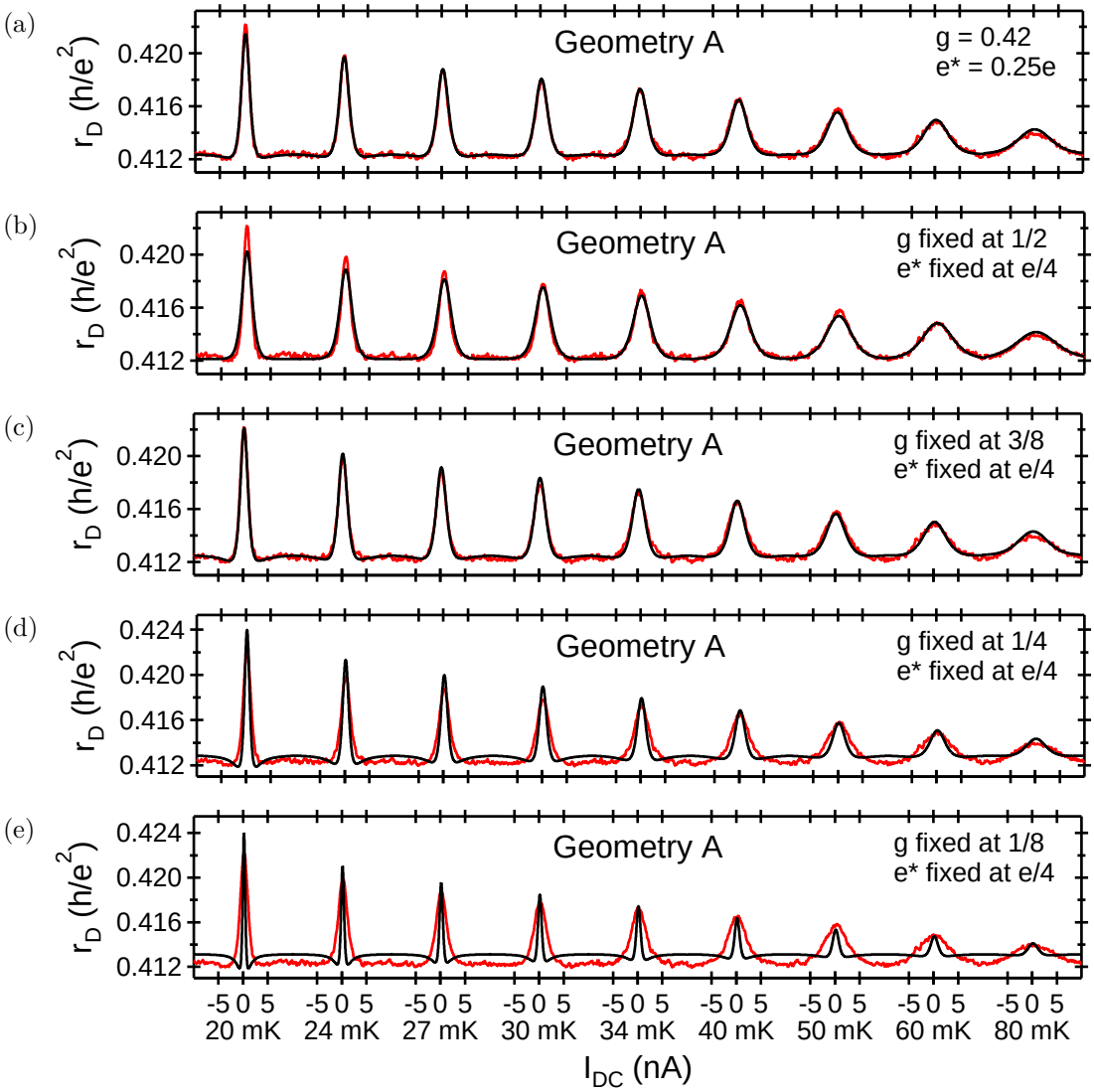


Figure 3.11: **Fits of the theoretical form of quasiparticle weak tunneling to r_D for Geometry A.** (a) Best fit, allowing e^* and g to vary. (b)-(e) Fits with e^* fixed at $e/4$ and g fixed at values corresponding to various proposed wave functions. In each case, all temperatures shown are fit simultaneously with the same fit parameters. Experimental results are red and fits are black. Fits for $g = 1/2$, $1/4$, and $1/8$ do not follow the peak height scaling with temperature nor do they match the minima to the sides of the zero-bias peak.

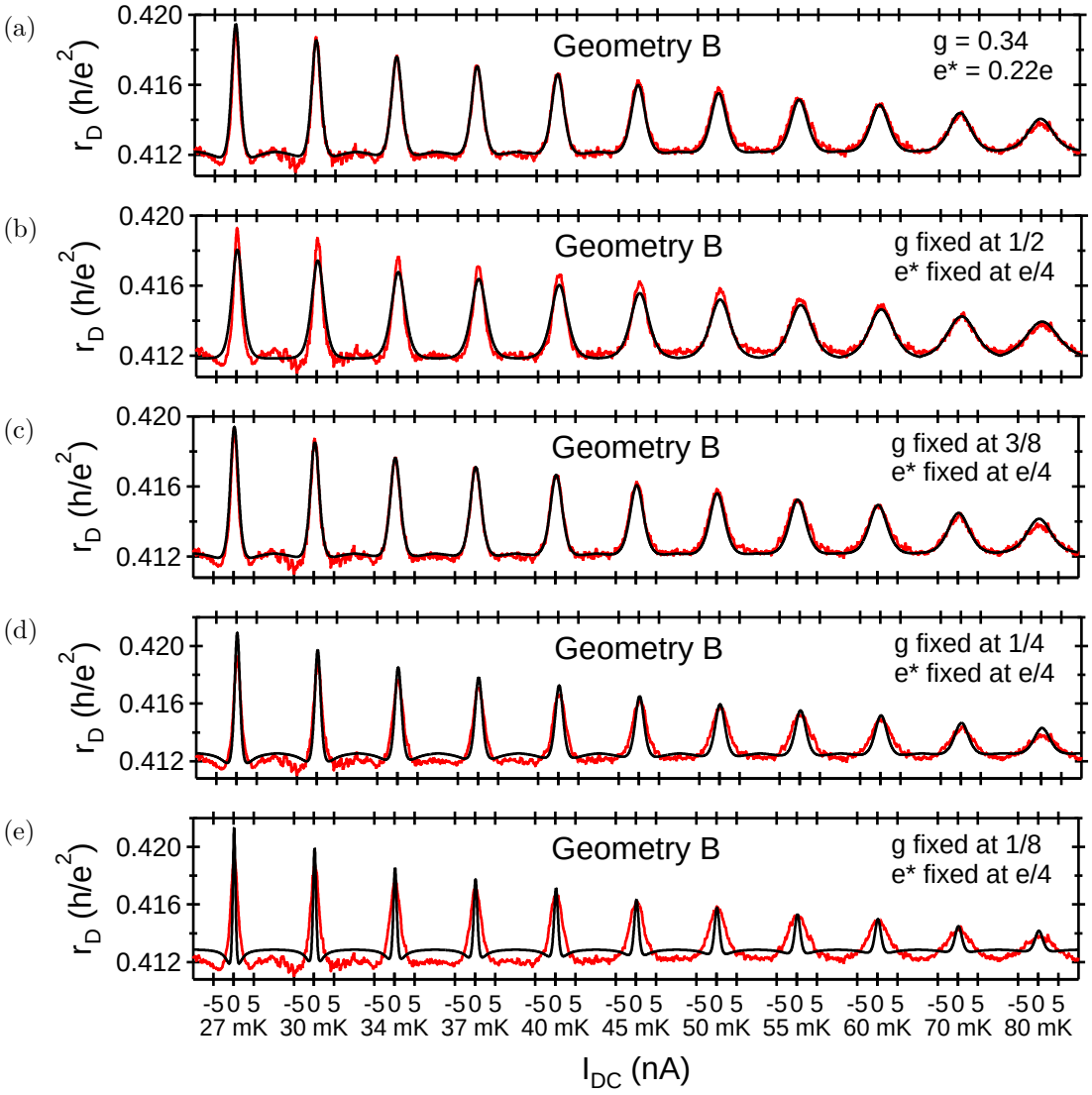


Figure 3.12: **Fits of the theoretical form of quasiparticle weak tunneling to r_D for Geometry B.** (a) Best fit, allowing e^* and g to vary. (b)-(e) Fits with e^* fixed at $e/4$ and g fixed at values corresponding to various proposed wave functions. In each case, all temperatures shown are fit simultaneously with the same fit parameters. Experimental results are red and fits are black. Fits for $g = 1/2$, $1/4$, and $1/8$ do not follow the peak height scaling with temperature nor do they match the minima to the sides of the zero-bias peak.

[50] results from a factor of about two reduction in the noise. Reaching a lower level of noise is particularly important because it makes the DC bias minima, which are important qualitative features of Equation (3.3), clearly distinguishable from the noise. The minima are even more prominent at base temperature, as can be seen in Figure 3.6.

In fact, the presence of these minima, which have also been observed in previous strong tunneling measurements [50] (also see Section 3.1), is an indication that g is strictly less than 1/2. Expanding the exact (but not closed form) expression for the tunneling conductance around the high DC bias limit [61, 83] one finds

$$g_t = C\left(\frac{T_B}{V_t}\right)^{2(1-g)}(2g-1) + \cdots . \tag{3.5}$$

Here, C is a positive constant, T_B is a temperature scale reflecting the strength of the edge channel interaction in the QPC, and V_t is the tunneling voltage. As T_B/V_t increases (corresponding to $|I_{DC}|$ decreasing from infinity), g_t decreases for $g < 1/2$, is constant for $g = 1/2$, and increases for $g > 1/2$. Since g_t eventually increases as the bias approaches zero, this produces a minimum in g_t (and similarly in r_d) only for $g < 1/2$. This behavior is also reflected in the weak tunneling formula, Equation (3.3), which is a perturbation result to first order in tunneling strength. This analysis neglects higher order terms, which could conceivably produce a minimum in g_t at $g = 1/2$, but to lowest order the presence of minima requires $g < 1/2$.

It may be that the value of g extracted from Equation (3.3) does not directly reflect the nature of the 5/2 wave function in our system. Reconstruction of edge channels [41] and interactions between edge channels may cause the value of g observed by tunneling experiments to change [84, 85]. The presence of striped phases,

which have been found to be energetically favorable for some confinement potentials [66], or random domains of different ground state wave functions [64] may also complicate matters. Even the chiral Luttinger liquid theory of quasiparticle tunneling may be an incomplete description of experimental systems. Experimental results of electron tunneling in cleaved edge overgrowth samples are not fully described by the predictions of chiral Luttinger liquid theory, indicating that perhaps a more nuanced picture is needed to accurately describe tunneling experiments [47]. Measurements of transmission through QPC devices in the lowest Landau level provide some confidence that chiral Luttinger liquid theory can be applied to quasiparticle tunneling across a QPC [62, 63, 86]. Ideally, one would like to repeat the measurement technique of Radu et al. [50] at a simple filling factor, such as $\nu = 1/3$, and fit Equation (3.3) to the results. Unfortunately, the sample studied here is not suitable for such measurements, because of the relatively wide width of the QPC and the low electron density.

In conclusion, fits based on quasiparticle weak tunneling theory [1] favor the presence of the 331 abelian wave function in our sample, while excluding other states, including the non-abelian Pfaffian and anti-Pfaffian. Particularly telling is the presence of minima in the DC bias dependence, which requires $g < 1/2$. Given previous studies favoring the Pfaffian and anti-Pfaffian wave functions [67, 70, 72, 76, 78] or a non-abelian wave function in general [11, 12], it seems possible that different states may be physically realizable at $\nu = 5/2$. The device geometry and heterostructure characteristics may be important factors in determining which state is favored. For example, there is numerical evidence that the strength of the confinement potential influences the wave function exhibited in a fractional quantum Hall system at $\nu = 5/2$

[66]. Better understanding of the effects of these factors on the 5/2 state will likely be vital for any efforts to further explore non-abelian particle statistics or realize a topological quantum computer. We recommend that similar measurements of e^* and g be performed on other heterostructures and device geometries, especially the ones in which evidence of a non-abelian wave function has been observed [11, 12].

Chapter 4

Breakdown at Various Filling Factors

The integer and fractional quantum Hall states are known to break down at high DC bias, exhibiting deviation from the ideal incompressible behavior. In this chapter we present measurements of breakdown of the $\nu = 2$, 3, 4, 5 integer and the $\nu = 4/3$ and 5/3 fractional quantum Hall states in a QPC of lithographic width $\sim 600\,\mathrm{nm}$. Breakdown of the QHE can be characterized by a well-defined nonzero critical current. We measure the dependence of the critical current on magnetic field, QPC gate voltage, and QPC width. Of particular interest, the critical current of the 4/3 and 5/3 fractional states shows the opposite dependence on QPC width compared to the integer states. This previously unobserved result is not explained by current theories of breakdown.

As discussed in Section 1.3, electrons confined to two dimensions with a perpendicular magnetic field exhibit the integer QHE at low temperatures [15], when the

thermal excitation becomes much smaller than the Landau level energy gap. In such a system, when the Fermi energy lies between Landau levels the longitudinal resistance R_{XX} vanishes while the Hall resistance becomes quantized at

$$R_{XY} = fh/e^2, \tag{4.1}$$

where f is equal to the number of filled Landau levels. The state of the system is characterized by the filling factor $\nu = nh/eB$, which takes integer values for the integer QHE. The fractional QHE [18], presented in Section 1.4, arises when electrons form composite fermions by binding to flux quanta [25]. These composite fermions in turn exhibit their own version of the QHE at fractional filling factors with odd denominators. Similar to the integer QHE, the longitudinal resistance vanishes and the Hall resistance obeys Equation (4.1), with modification that f takes fractional values. However, unlike the integer QHE, the fractional version is fundamentally a result of collective interactions of electrons.

In both the integer and fractional regimes, quantized quantum Hall states will not be observed if electrons or quasiparticles are able to access extended states other than the ground state. This can happen, for example, if the temperature becomes comparable to the energy gap of the system [87]. Alternatively, a DC bias will cause breakdown of the quantum Hall states if the current through the sample exceeds some critical value [88, 89]. At currents larger than the critical value the longitudinal resistance sharply increases above its zero-bias value (which vanishes as temperature goes to zero). Multiple studies of this type of breakdown have been performed, mostly focused on the integer quantum Hall regime; see the review by Nachtwei for examples of earlier work [14]. Breakdown has been observed in the fractional quantum Hall

regime [90, 91], but to our knowledge no dependence of the critical current on sample width has been reported.

The value of the critical current is sample-dependent and has been found for the integer QHE to increase either linearly [92–99] or sublinearly [96–100] with increasing sample width. Two explanations, each relating to the scale of inhomogeneities in the sample, have been proposed [14, 96, 98, 99]. The inhomogeneities, which arise from potential fluctuations due to ionized Si donors and to impurities or imperfections in the sample, are believed to result in various regions of compressible and incompressible states [101]. In this model, the QHE is observed when the corresponding incompressible state percolates throughout the sample.

We first consider an explanation for the two types of width dependence which emphasizes the length scale of the inhomogeneities [14]. If the width of the sample is much larger than this characteristic scale, then a large number of percolation paths will be present and this number will scale roughly linearly with the sample width. If each percolation path is assumed to contribute to the total current then the (total) critical current will increase linearly with the number of paths, and hence with sample width. Conversely, if the width of the sample is of the same order as or smaller than the inhomogeneities, then the critical current is believed to scale nonlinearly with sample width.

The alternative explanation that has been proposed considers the energy scale of the inhomogeneities relative to that provided by temperature k_BT [98, 99]. It assumes that a large number of percolation paths are present in the sample. At low temperatures, such that k_BT is less than the characteristic energy of the inhomogeneities,

the picture is the same as above—the critical current depends linearly on the number of percolation paths and hence on the width of the sample. However, at high temperatures, such that k_BT exceeds the characteristic energy, the thermal broadening washes out the inhomogeneities and the sample becomes essentially uniform. A calculation for an ideal homogeneous sample based on the calculated self-consistent potential [102] indicates that the critical current should increase sublinearly with sample width [100]. However, other authors have called the underlying assumptions of this calculation into question [14, 103].

A number of explanations for the breakdown as a whole have been proposed, but no consensus seems to have been reached [14]. We briefly discuss a number of these theories, focusing on the behavior predicted as a function of sample width. Eaves and Sheard propose that breakdown is caused by quasi-elastic inter-Landau-level scattering (QUILLS) [104]. In this process, an electron from the uppermost filled Landau level scatters into an empty state of equal energy in the next, unfilled, Landau level. Normally such scattering is highly suppressed due to the energy gap between Landau levels; however if the Hall (transverse) electric field is strong enough, two such states of equal energy will have enough overlap in their wave functions to allow scattering to occur. In this model, breakdown occurs at a critical Hall field, and hence the critical current increases with sample width. Shizuya considers what he refers to as an "intrasubband process" in which an increased Hall field causes delocalization of previously-localized impurity states [105]. The newly extended states carry current through the sample, leading to breakdown. Again, because of the role of the Hall field, the critical current is expected to increase with sample width. A number of

authors have considered electron heating as a cause of breakdown [106–108]. Energy is gained by the electric field, with a power $P = \mathbf{J} \cdot \mathbf{E} \approx \sigma_{XX} E_H^2$, and dissipated by some (generally unspecified) mechanism. Reasonable assumptions for the dependence of the dissipation on the electron and lattice temperatures leads to runaway heating, which in turn causes breakdown, at a critical Hall field. Střěda and von Klitzing consider an electron-phonon interaction, in which an electron moves from one edge of the sample to the other, with the excess energy absorbed by phonons [109]. This is allowed only when the drift velocity $v = E_H/B$ of electrons exceeds the sound velocity in GaAs. As with previous theories, this implies the existence of a critical Hall field. Finally, Tsemekhman et al. consider a percolation model, as described above [101], and calculate the critical Hall field at which a metallic path opens across the sample [110]. This metallic path is caused by an avalanche-type breakdown of successive incompressible regions, leading to breakdown of the sample as a whole.

Each of these theories explain in their own way the observation of increasing critical current with increasing sample width. However, we find that in the fractional quantum Hall regime ($\nu = 4/3$ and $5/3$) the critical current decreases with increasing sample width. This is qualitatively opposite as the behavior observed in the integer quantum Hall regime. Current theories of QHE breakdown are unable to account for this result. This may indicate that the theories are merely incomplete, or that they are fundamentally flawed. In either case, it seems that our understanding of breakdown of the QHE is far from complete.

Understanding the mechanism by which quantum Hall states break down is important for a few reasons. First, understanding the causes and mechanism of breakdown

will likely improve understanding of the QHE itself. Second, understanding the limits of validity of the ideal QHE picture presented in Chapter 1 is important, especially for design of experiments and interpretation of experimental data. Nonzero temperature already imposes one important limitation to the ideal picture, as discussed in Section 1.3. Breakdown imposes another type of limitation, which is likely more detrimental to experiments but, luckily, is easier to avoid. Many experiments use the same technique of defining the sample geometry with metallic gates that we discuss in Section 2.1 and the QPC is one of the common building blocks of these devices [6, 7, 20, 21]. Thus it is particularly important to understand breakdown in QPCs and these types of devices in general. As a simple example, one must carefully limit the DC bias in these experiments to avoid accidentally moving into the breakdown regime. Finally, the highly quantized Hall resistance in the integer QHE is currently used as the international standard of resistance [111]. Naively, one would like to measure this with large currents, to improve the signal to noise ratio, but too large a current threatens to lead to breakdown. Understanding under what conditions breakdown occurs and the resulting effects on the Hall resistance is important to improve and maintain the standard of resistance.

We present measurements of a QPC formed in the high-mobility GaAs/AlGaAs heterostructure described in Section 2.1 and discussed throughout the rest of Chapter 2. At base temperature the sample mobility is measured to be $1 \times 10^7\,\mathrm{cm^2\,V^{-1}\,s^{-1}}$ and the electron density to be $2.6 \times 10^{11}\,\mathrm{cm^{-2}}$. In the middle of the Hall bar a QPC is created by applying a voltage to metallic gates on the surface of the chip. Applying a negative voltage to these gates depletes the electrons from the 2DES underneath,

creating a constriction. Gates A1 and G3 of Figure 2.2 are energized to form the QPC, while the remaining four gates are kept grounded during the experiment. The lithographic width of the QPC is approximately 600 nm. The sample is cooled in a dilution refrigerator with a base electron temperature of ~ 15 mK.

A schematic of the mesa and measurement setup is shown in Figure 2.4. An AC current of 0.4 nA RMS is sourced on one end of the Hall bar and the other end is grounded. Measurements of differential resistances r_{XX}, r_{XY}, and r_D (as shown) are performed using standard lock-in techniques at 17 Hz. The local physics of the QPC are probed by r_D, which is measured across the Hall bar and on opposite sides of the QPC. As discussed in Section 2.1, r_D is very nearly the same as r_D^+, which was analyzed using the Landauer-Büttiker formalism in Section 1.5. Conversely, r_{XX} and r_{XY} are insensitive to the QPC and, instead, depend on the large-scale physics of the Hall bar far away from the QPC. Breakdown is studied by applying a DC current I_{DC} of up to ± 100 nA. Previous studies of breakdown of quantum Hall states have mostly focused on measurements of the longitudinal resistivity ρ_{XX}, or similar quantities, measured using voltage contacts on the same side of the Hall bar but on either side of the region of breakdown. This differs from our r_{XX}, which is measured far away from the QPC and reflects the behavior of the wide Hall bar. However, our measurements are performed in regimes in which r_{XY} is well-quantized and constant. Hence, using Kirchhoff's voltage law, the quantity $r_D - r_{XY}$ should exhibit similar behavior to the ρ_{XX} measured by others.

We observe breakdown of the quantum Hall states in the QPC at DC biases for which r_{XX} and r_{XY} show no sign of breakdown outside the QPC. However, in order

for this comparison to be meaningful, the electron density in the region inside the QPC needs to be equal to the density far away in the Hall bar. Because the magnetic field is constant throughout the sample, a uniform electron density will produce a uniform filling factor, allowing for valid comparison of r_D with r_{XY} and r_{XX}. As discussed in Section 2.2, energizing the QPC gates normally reduces the electron density in the region of the QPC to less than that in the surrounding Hall bar. For example, applying gate voltages similar to those used in this study in the usual way at base temperature would reduce the electron density from 2.6 to $2.2 \times 10^{11}\text{cm}^{-2}$, more than a 15% reduction. This is illustrated in Figure 2.7. We avoid this problem by using the technique of annealing the gates at 4 K [50], which is discussed in detail in Section 2.2. A gate voltage of $-2.7\,\text{V}$ is applied at 4 K for ~ 40 hours before cooling to base temperature. Figures 2.5 and 2.8 demonstrate that this annealing technique is effective in maintaining a uniform electron density throughout the sample. At $V_G = -2.7\,\text{V}$ the density in the QPC is slightly less than in the full Hall bar, but significantly closer than the 15% difference found without annealing. This small difference is compensated for by going to more positive gate voltage $V_G \geq -2.4\,\text{V}$, at which point the electron density is essentially uniform throughout the sample. Measurements of breakdown are performed in the range $-2.4\,\text{V} \leq V_G \leq -2.0\,\text{V}$. In summary, annealing creates, for the gate voltages used in our measurements, an electron density in the QPC within 1% of the density elsewhere throughout the Hall bar.

The breakdown feature we observe is shown in Figure 4.1a. Differential resistances r_D, r_{XY}, and r_{XX} versus I_{DC} are measured simultaneously at various magnetic fields

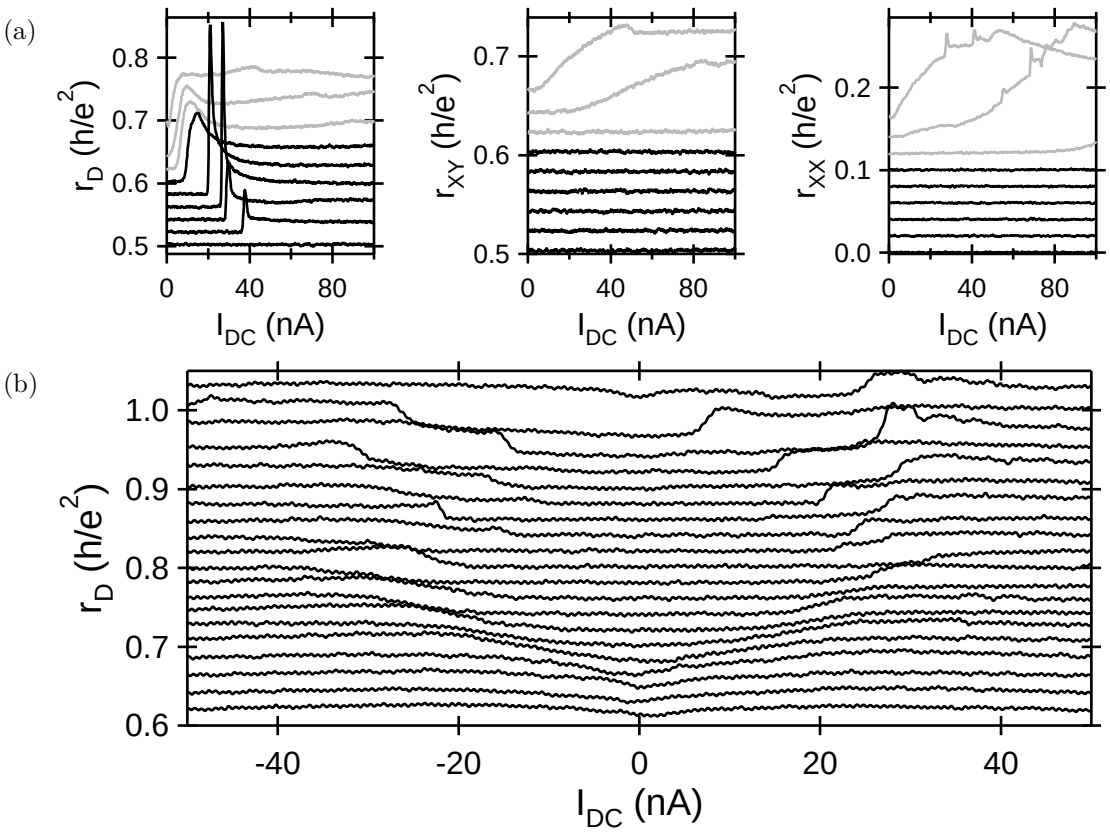
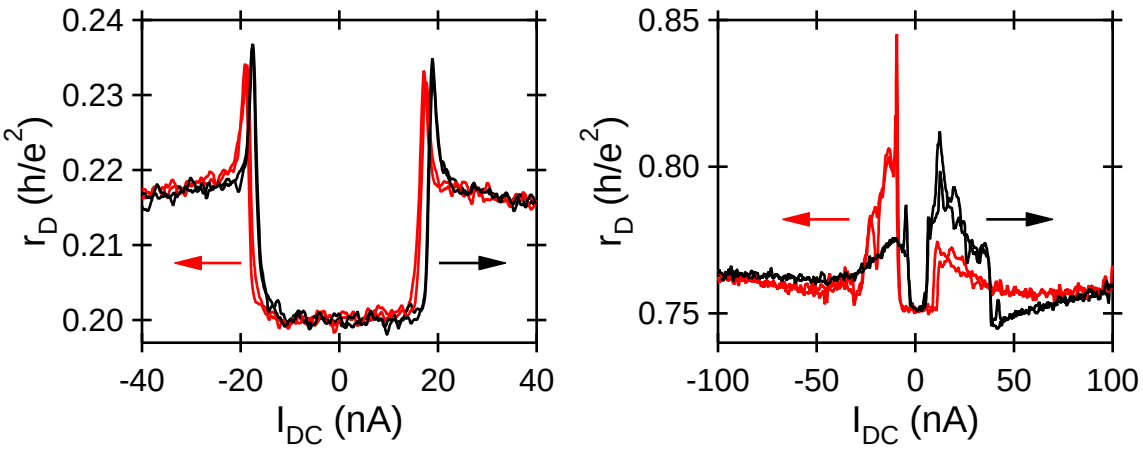


Figure 4.1: **Example of breakdown in the QPC at nonzero critical currents.** (a) Simultaneous measurements of r_D, r_{XY}, and r_{XX} near $\nu = 2$. All curves are measured with increasing DC bias. Consecutive scans, from bottom to top, are separated by 80 mT (5420 to 6060 mT) and offset by $0.02\,h/e^2$ vertically. Black curves are those for which r_{XY} remains quantized and r_{XX} is zero within the noise over the entire range of DC bias. Grey curves are those for which r_{XY} and/or r_{XX} deviate from these values. Breakdown of the quantum Hall states in the QPC is characterized by a sharp increase in r_D at the critical current. (b) Breakdown of the $\nu = 5/3$ plateau. Consecutive scans, from bottom to top, are separated by 20 mT (6300 to 6680 mT, covering slightly more than the 5/3 plateau) and offset by $0.02\,h/e^2$ vertically. DC bias is swept in alternating directions for consecutive scans, with increasing I_{DC} for the bottom scan.

on the high magnetic field side of the $\nu = 2$ plateau. On the plateau (black curves), r_{XY} is well-quantized and $r_{XX} = 0$ within the noise. As B is increased past the plateau (grey curves) both r_{XY} and r_{XX} show deviations from this behavior. In contrast, r_D increases from the quantized value at lower magnetic fields, showing an abrupt rise with, in many cases, a sharp peak and a high-DC-bias value larger than the quantized plateau resistance. Because we are measuring the differential resistance, the sharp peak corresponds to a steep rise in the current above threshold. The critical current I_c, at which the sharp rise in differential resistance occurs, is generally well-defined and we use the value of I_{DC} at which r_D rises above the noise on the plateau. Figure 4.1b shows similar measurements of r_D over the entire $\nu = 5/3$ plateau. Here the breakdown does not exhibit a peak and the rise off of the quantized plateau is not as sharp. However, we can still accurately define the critical current as the point at which r_D exceeds the noise on the plateau.

We measure I_c as a function of magnetic field within each plateau and of gate voltage at fixed magnetic field. In each case we sweep I_{DC} between $\pm 100\,\mathrm{nA}$, alternating directions, while stepping B or V_G after each sweep. For each value of B or V_G we calculate $|I_c|$ as described above; the spread in $|I_c|$ at each point is predominantly the result of hysteresis in the I_{DC} sweep direction. We observe two different types of hysteresis, each of which seems to occur inconsistently. One is similar to that observed in previous experiments [89, 98, 112], illustrated in Figure 4.2a. We also observe a strange type of hysteresis illustrated in Figure 4.2b. In some cases, both types of hysteresis occur simultaneously. For all measurements, the I_{DC} sweep rate is $0.7\,\mathrm{nA/s}$. We have tested the dependence on sweep rate at filling factors 2 and 5,



(a) (b)

Figure 4.2: **Examples of two different types of hysteresis.** (a) Hysteresis in which I_c increases with increasing I_{DC} and decreases with decreasing I_{DC}. (b) A different type of hysteresis in which $|I_c|$ is different for the two sweep directions. For both graphs, the curves are measured by continuously sweeping I_{DC} in alternating directions and incrementing gate voltage by 4 mV between sweeps; arrows indicate the sweep direction by color. Curves in (a) are taken at 2.26 T ($\nu = 5$) with gate voltages -2.236 to -2.224 V. I_{DC} is swept between ± 100 nA but only displayed over a limited range to emphasize the hysteresis. Sweeps in (b) are taken at 8.15 T ($\nu = 4/3$) with gate voltages -2.280 to -2.268 V.

sweeping I_{DC} with fixed B and V_G, and find no significant differences for slower sweep rates down to $\sim 1 \times 10^{-3}$ nA/s. These sweep rates tests are performed by alternating the sweep direction and stepping the rate after each sweep. This is very similar to the procedure used to measure the dependence of r_D on B and V_G. However, we observed no hysteresis during the sweep rate tests; in other words, each sweep looks the same, independent of sweep rate or direction. This may indicate that hysteresis only occurs for faster sweep rates (around 0.7 nA/s). On the other hand, this is not conclusive, as even at sweep rates of 0.7 nA/s hysteresis is not always observed.

Critical currents for integer and fractional filling factors are shown in Figure 4.3 as a function of B and for two values of V_G. For integer filling factors, I_c is strongly nonlinear with magnetic field. Similar nonlinearities have been observed in GaAs samples of width ~ 1 micron [113, 114] and 380 microns [89] at $\nu = 2$, and in graphene [115], although other studies have observed different (mostly linear) behavior [116–119]. The dependence on magnetic field exhibits power law behavior, at least over a limited range. Least-squares fits are performed using a power law equation

$$|I_c| = A(B_0 - B)^{\alpha} \tag{4.2}$$

with B_0 fixed to the lowest magnetic field at which r_D is not quantized at zero DC bias. Resulting fits are displayed in Figure 4.4 and the corresponding best-fit values of the power α are listed in Table 4.1. See the caption of Figure 4.4 for more details about the fitting procedure.

As B is decreased towards the center of each plateau, $|I_c|$ increases beyond the 100 nA range over which we measure. Although we are still able to measure $|I_c|$ near to the center of the plateau, the strong nonlinearity of I_c makes it hard to estimate

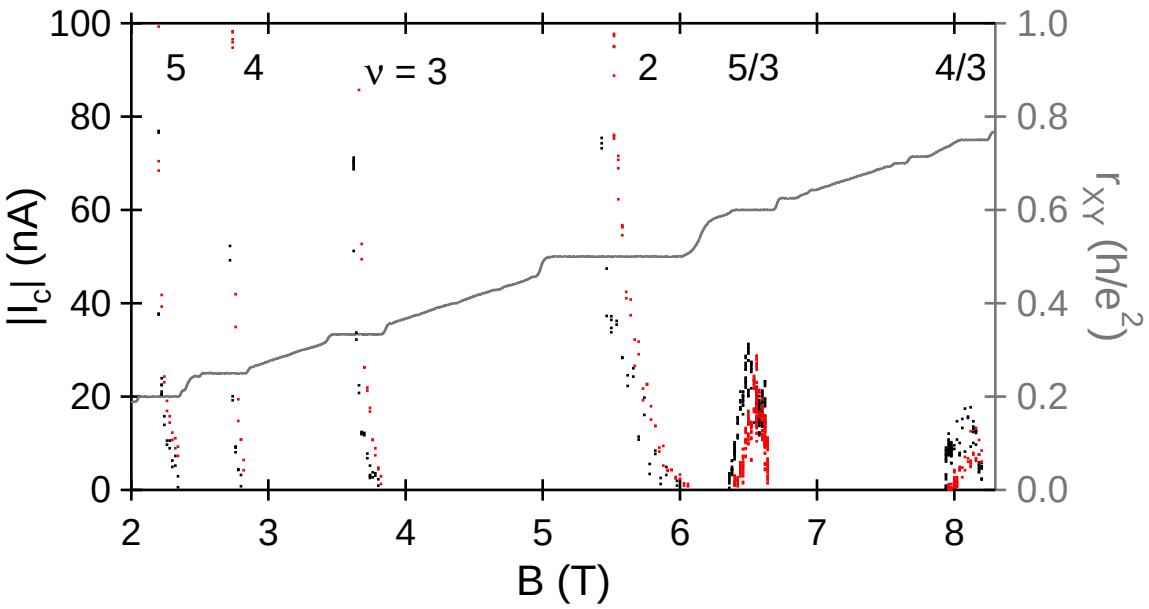


Figure 4.3: **Critical current as a function of magnetic field for various filling factors.** Absolute value of the critical current plotted against magnetic field over the $\nu = 4/3$, $5/3$, 2, 3, 4, and 5 quantum Hall plateaus. Black dots are measured with a $-2.4\,\mathrm{V}$ gate voltage and red dots with $-2.2\,\mathrm{V}$. The light grey curve (right axis) shows a r_{XY} Hall trace taken separately for reference. As explained in the text, breakdown on the low magnetic field side of the integer plateaus is qualitatively different and the critical current in these regions is not shown.

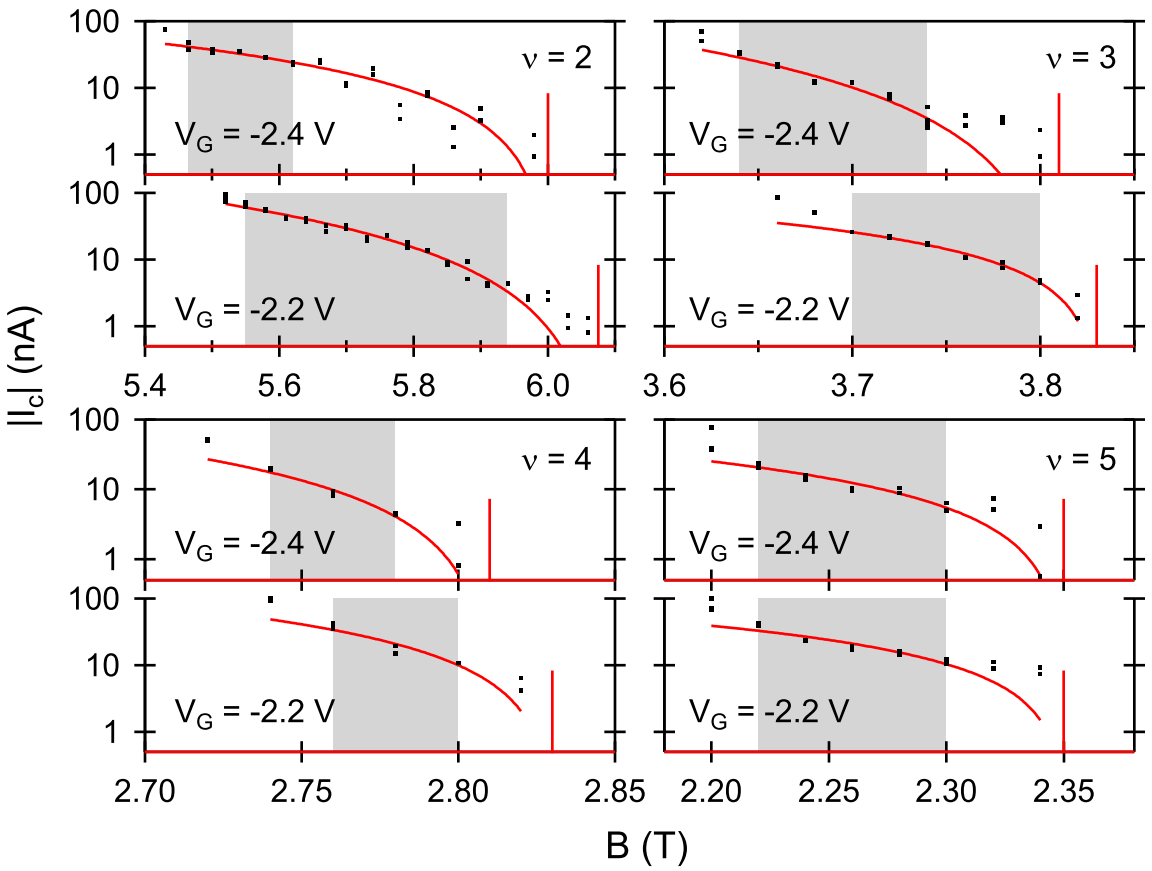


Figure 4.4: **Power law fits of critical current dependence on magnetic field.** Least-squares fits of Equation (4.2) on the high-field side of integer quantum Hall plateaus. Four filling factors are studied, each at two gate voltages, as labeled. Black dots are values of the critical current measured by sweeping I_{DC} at fixed magnetic field, which is incremented by ΔB after each sweep. Red curves are best fit results. The magnetic field offset B_0, at which $|I_c| \to 0$, is fixed to a value $\Delta B/2$ greater than the highest magnetic field at which a nonzero critical current is measured. These values of B_0 are indicated by the vertical red lines. Because the measurements deviate from power law behavior at the highest and lowest magnetic fields, fits are only performed over the regions shaded grey, and the results extrapolated to cover the full range. Error bars given in Table 4.1 are obtained by repeating the fits with B_0 changed by $\pm\Delta B/2$.

Table 4.1: Exponents extracted by power law fits using Equation (4.2) at integer filling factors.

	α	
ν	$V_G = -2.4\,\mathrm{V}$	$V_G = -2.2\,\mathrm{V}$
2	1.6 ± 0.2	2.1 ± 0.1
3	2.4 ± 0.2	1.2 ± 0.2
4	1.7 ± 0.4	1.4 ± 0.2
5	1.4 ± 0.2	1.2 ± 0.2

the maximum value of $|I_c|$, which presumably occurs at the center of each plateau. We do also observe deviations of r_D from the quantized value on the low magnetic field side of integer plateaus. Here r_D falls below the quantized value at high $|I_{DC}|$, as illustrated in Figure 4.5. In addition, we find that r_{XY} and r_{XX} also show deviations at nearly the same critical currents. Furthermore, compared to the high-field side of the plateaus, $|I_c|$ increases even more rapidly with changing B. On the low-field side the portion of the plateau for which $|I_c| < 100\,\mathrm{nA}$ is only 20% or less of the plateau width, compared to $\sim$ 50% of the plateau on the high-field side. It appears that on the low magnetic field side of integer quantum Hall plateaus, the QPC and the full Hall bar undergo similar transitions out of the incompressible state. This is in contrast to the high-field side of the plateaus where $|I_c|$ is clearly lower for r_D than for r_{XY}. Hence we do not plot $|I_c|$ values for the low field side of the integer plateaus in Figure 4.3.

We observe breakdown at fractional filling factors of 4/3 and 5/3, but not at 5/2, which instead exhibits a zero-bias peak due to backscattering at the QPC [50]

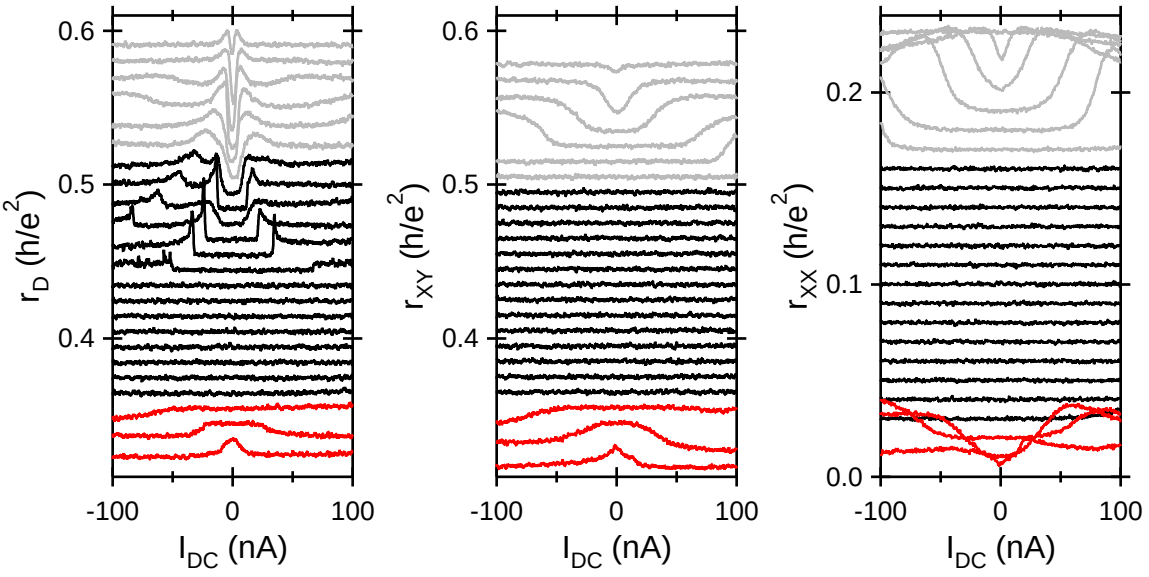


Figure 4.5: **Example of breakdown at an integer quantum Hall plateau.** Differential resistances r_D, r_{XY}, and r_{XX} measured simultaneously as a function of DC bias around the $\nu = 3$ plateau. Consecutive scans, from bottom to top, are separated by 20 mT (3400 to 3840 mT) and offset by $0.01\,h/e^2$ vertically. DC bias is swept in alternating directions for consecutive scans, with decreasing I_{DC} for the bottom scan. Black curves are those for which r_{XY} remains quantized and r_{XX} is zero within the noise over the entire range of DC bias. Grey curves are those for which r_{XY} and/or r_{XX} deviate from these values on the high magnetic field side of the plateau. The corresponding measurements of r_D exhibit breakdown at much smaller critical currents. In contrast, on the low field side of the plateau all three resistances exhibit deviations from ideal behavior at similar values of DC bias and magnetic field (red curves).

(also see Chapter 3). The behavior of breakdown at fractional filling factors 4/3 and 5/3 is different from that at the integers in three ways. First, $|I_c| \leq 100\,\mathrm{nA}$ over the entire plateau, allowing us to measure the maximum $|I_c|$ near the middle of the plateau. Second, r_D qualitatively maintains the same behavior (r_D increases for $|I_{DC}| > |I_c|$) along the entire plateau. This is in contrast to the integers, which show qualitatively different behavior on the high and low magnetic field sides, as shown in Figure 4.5. Last, $|I_c|$ exhibits the opposite dependence on V_G, as indicated by the black ($V_G = -2.4\,\mathrm{V}$) and red ($V_G = -2.2\,\mathrm{V}$) curves. For integer filling factors, more positive gate voltage leads to greater $|I_c|$, while for the fractions $|I_c|$ decreases or remains the same.

We can compare the maximum critical currents in the fractional plateaus to values measured by others. Takamasu et al. find a relationship between the critical currents at various integer and fractional filling factors:

$$I_c \propto \left|\frac{B_{\mathrm{eff}}}{B}\right|, \tag{4.3}$$

with B_{eff} the effective magnetic field [90]. For integer states $B_{\mathrm{eff}} = B$. However, the composite fermions which form the fractional QHE states experience a reduced magnetic field [120]; for the $\nu = 4/3$ and 5/3 states $B_{\mathrm{eff}} = |B - n\Phi_0/(3/2)|$. Equation (4.3) implies that the 4/3 and 5/3 states would exhibit the same critical current. However, we find that $|I_c|$ is significantly larger for $\nu = 5/3$, in contradiction to the results of Takamasu et al. [90].

Figure 4.6 more directly exhibits the dependence of $|I_c|$ on gate voltage at fixed magnetic field for each filling factor. Each of the four integer states exhibit increasing $|I_c|$ with more positive V_G, while the fractional states have the opposite dependence.

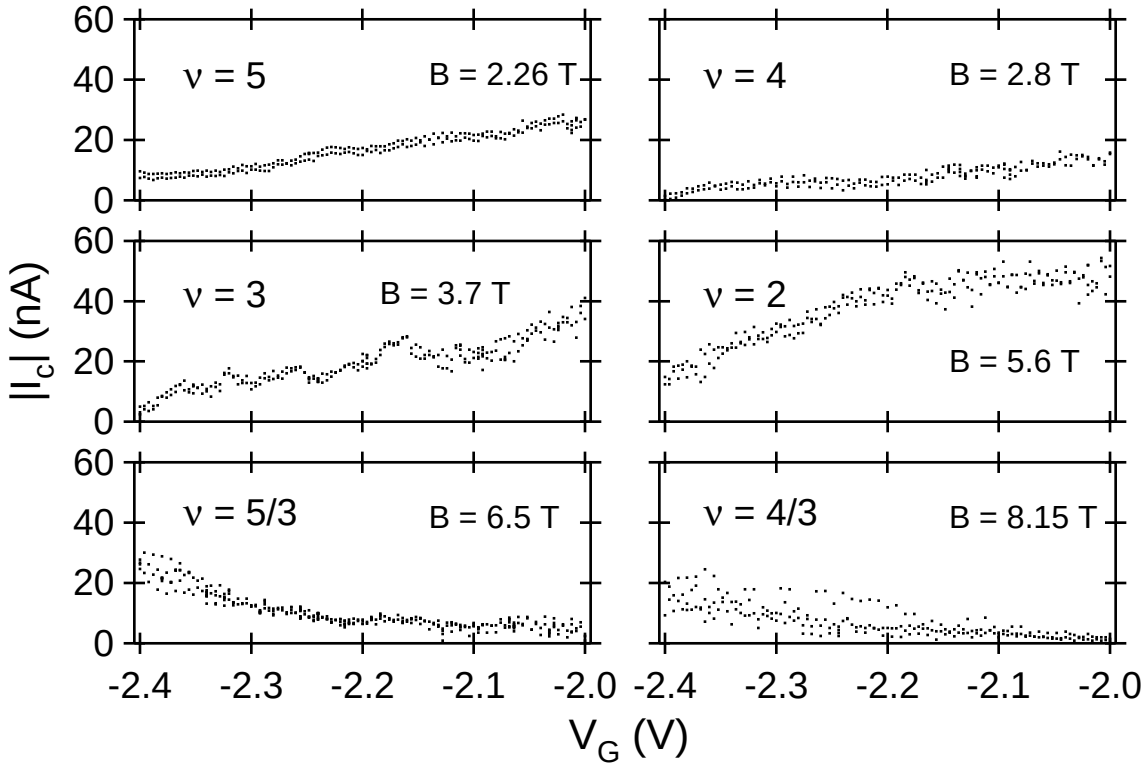


Figure 4.6: **Critical current as function of gate voltage at various filling factors.** Absolute value of the critical current plotted against gate voltage for $\nu = 4/3$, 5/3, 2, 3, 4, and 5. Each curve is measured at the indicated magnetic field. Integer filling factors exhibit critical currents increasing with more positive gate voltage while fractional filling factors exhibit decreasing critical currents.

Changing the gate voltage has two possible consequences: changing the electron density in the QPC and changing the width of the QPC. As discussed in detail in Section 2.3, we are able to estimate the QPC width and electron density as a function of V_G using the behavior of r_D near zero magnetic field. Unlike r_{XY}, r_D is nonzero at zero field; the increased resistance is caused by the QPC constriction. The combination of the electrostatic confinement and magnetic field leads to the formation of magnetoelectric subbands which are depopulated with increasing magnetic field [46]. For different potential profiles, one can calculate the resistance through an infinitely long channel as a function of B [46, 53]. In Section 2.3 we fit two different models to the low-field r_D data to extract the QPC width and the electron density in the QPC. Results of the two approaches are shown in Figure 2.8, along with the electron density in the Hall bar calculated from the slope of r_{XY} versus B. Although the electron density in the QPC changes slightly with V_G, the change is not monotonic and the electron density in the QPC remains within 1% of that of the Hall bar over the range of gate voltages used in this experiment ($V_G \geq -2.4\,\mathrm{V}$). The dependence of I_c on V_G observed for the fractional filling factors cannot be explained by the measured changes in electron density. For fixed filling factor, a change in electron density would produce the same effect as a change in magnetic field. However, because the electron density changes nonmonotonically with V_G, such changes cannot explain the monotonic change in $|I_c|$. In addition, even in regions with qualitative agreement, the magnitude of the change in electron density falls short by a factor of ~ 2 to 10 relative to that needed explain the change in I_c.

In contrast, the QPC width shows a strong dependence on V_G. When $V_G = V_G^{\mathrm{anneal}}$,

the fits give widths in rough agreement with the $\sim 600\,\mathrm{nm}$ lithographic width of the QPC. Presumably, the annealing causes the channel width to be greater than it would otherwise be. As V_G becomes more positive, the width increases linearly to be on the order of a few microns. We perform a linear fit of QPC width versus V_G and use the results to convert the I_c dependence on V_G of Figure 4.6 to a dependence on QPC width. The results are shown in Figure 4.7, using the width given by the square well potential model. The $\nu = 3$, 4, 5 states exhibit a linear dependence of $|I_c|$ on the QPC width, while $\nu = 2$ appears perhaps sublinear. Extrapolating to $|I_c| = 0$ for integer filling factors gives a nonzero intercept on the width axis on the order of $1\,\mu\mathrm{m}$. Perhaps nonintuitively, we still observe a well-quantized QHE when the QPC width is less than $1\,\mu\mathrm{m}$ (e.g. $V_G = -2.7\,\mathrm{V}$), meaning that $|I_c| > 0$ even for widths less than those given by the intercepts in Figure 4.7. The value of the intercept is similar for the $\nu = 3$, 4, and 5 states, which is consistent with previous experiments [92, 94]. The fractional states show opposite behavior: $|I_c|$ decreases with increasing QPC width. This is the first reported result of width dependence of breakdown in the fractional quantum Hall regime of which we are aware, and is in direct contrast to all previously reported results in the integer regime.

During a different cool-down of the same device we measured the temperature dependence of breakdown of the $\nu = 2$ state. In this case the QPC is formed by annealing gates G1, G2, and A2 at $-3.0\,\mathrm{V}$ for $\sim$ 60 hours, with the other gates remaining grounded (see Figure 2.2). Similar to the data displayed above, we present $|I_c|$ versus B at $V_G = -2.7\,\mathrm{V}$ and $|I_c|$ versus V_G at $5.8\,\mathrm{T}$. The results are displayed in Figures 4.8a and 4.8b, respectively. In neither case do we observe any temperature

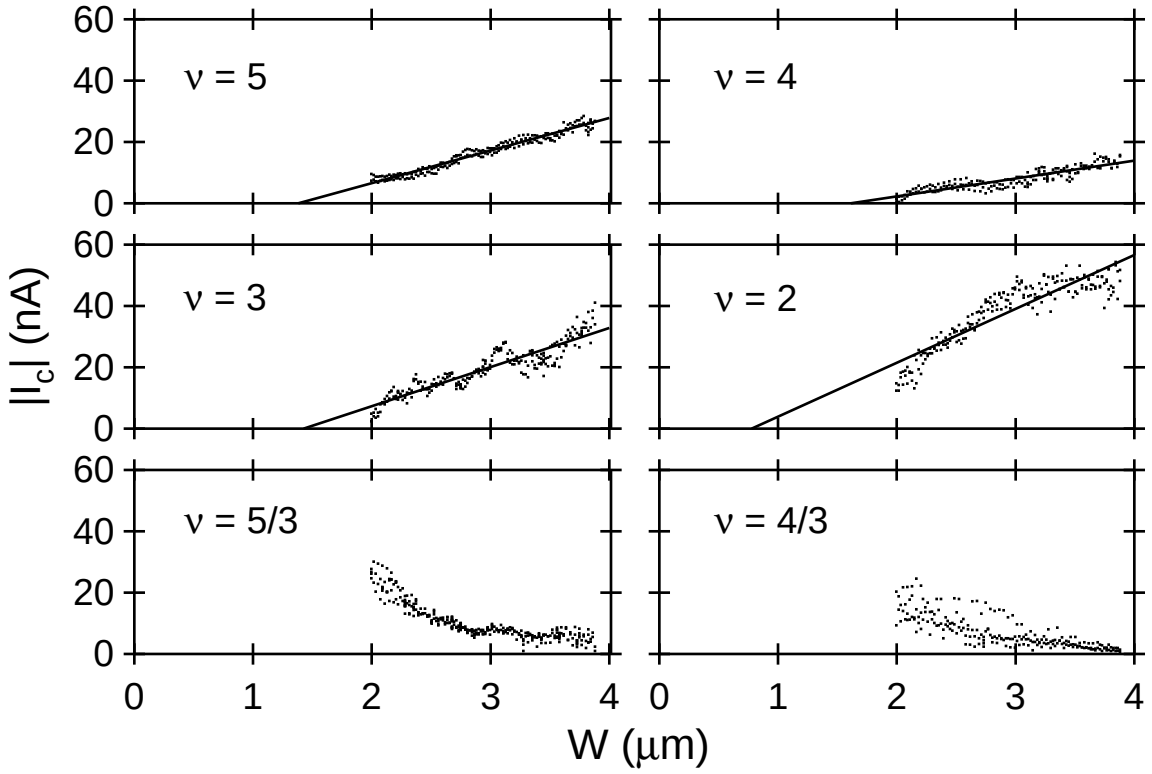


Figure 4.7: **Critical current as function of QPC width at various filling factors.** Results of Figure 4.6 plotted against QPC width at the labeled filling factors. Widths are estimated as a function of gate voltage using the square well potential model (see Section 2.3 and Figure 2.8). Results for the parabolic potential would increase all widths by $\sim 30\%$. Lines are linear fits at each integer filling factor.

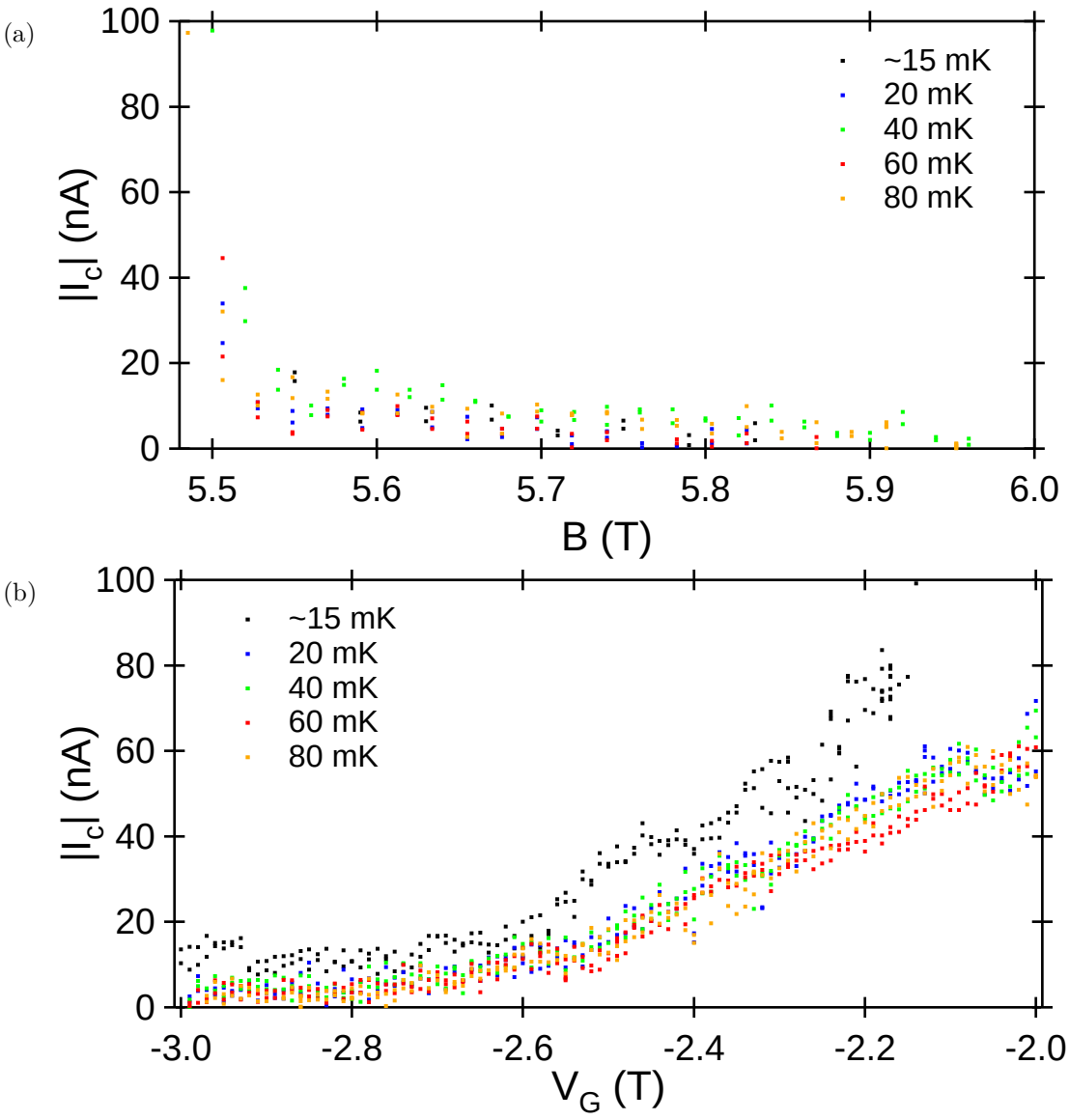


Figure 4.8: **Temperature dependence of critical current versus magnetic field and gate voltage.** (a) Critical current as a function of magnetic field at $\nu = 2$ and $V_G = -2.7\,\mathrm{V}$. (b) Critical current as a function of gate voltage at $B = 5.8\,\mathrm{T}$. Temperatures are electron temperatures, which track the mixing chamber temperature well down to 20 mK, and are estimated using techniques discussed in [53] below that point.

dependence between 20 and 80 mK. The measurement of I_c dependence on V_G at base temperature (corresponding to ~ 15 mK electron temperature) does exhibit an increased critical current compared to the other temperatures. This difference is fairly small (in absolute terms) at more negative gate voltages and increases as the gate voltage is made more positive. However, there is no difference in the critical current as a function of magnetic field at base temperature compared to 20 mK and higher. Given the consistency between the remainder of the measurements, it is possible that the 15 mK measurement of I_c versus V_G is an experimental outlier.

Appendix A

List of Symbols

A list of symbols and abbreviations, with the page number on which each is introduced and a brief description/definition.

Notation	Page	Description
2DES	35	two-dimensional electron system; refers to the active region of a GaAs/AlGaAs heterostructure
α	96	exponent in the power law equation (4.2) fit to critical current as a function of magnetic field
ΔB	98	step size in magnetic field between consecutive sweeps of I_{DC} when measuring critical current as a function of magnetic field
ϵ	14	permittivity of GaAs
θ	22	phase in the wave function due to particle exchange
μ_k	26	chemical potential; indexed by the contact k at which it is measured
μ_B	10	Bohr magneton
ν	6	Landau level filling factor
ρ_{XX}, ρ_{XY}	13	longitudinal and Hall resistivity, respectively
σ_{XX}	89	longitudinal conductivity
ϕ_j	5	normalized eigenfunction of the one-dimensional harmonic oscillator, indexed by quantum number j
Φ_0	6	quantum of magnetic flux; $\Phi_0 = h/e$
Ψ	5	single-particle electron wave function
ω_c	5	classical cyclotron frequency; $\omega_c = eB/m^*$

$\omega_{\rm par}$	49	characteristic frequency of the parabolic potential $V(x) = m^*\omega_{\rm par}^2 x^2/2$ used to model the QPC confinement potential
A	4	area of a rectangular cross-section perpendicular to current flow in a three-dimensional sample; $A = L_y L_z$
A	72	fitting parameter that sets the amplitude scale of the quasiparticle tunneling conductance formula
$\mathbf{A}$	5	magnetic vector potential, with components A_x, A_y, and A_z; we consider the Landau gauge $\mathbf{A} = -By\hat{\mathbf{x}}$
AC	41	alternating current; uses a sinusoidal waveform
AFM	120	atomic force microscope
$\mathbf{B}$	3	magnetic field vector, with magnitude B and components B_x, B_y, and B_z
B_0	96	magnetic field intercept in the power law equation (4.2) fit to critical current as a function of magnetic field
$B_{\rm eff}$	101	effective magnetic field experienced by composite fermions in the fractional quantum Hall regime
BCS	22	Bardeen, Cooper, and Schrieffer
C	82	a positive constant in an expansion of exact quasiparticle tunneling conductance
$\mathcal{C}$	20	an operator used in the Jain hierarchy; represents particle-hole conjugation
CF	19	composite fermion
CLL	28	chiral Luttinger liquid
CMSE	115	Center for Material Science and Engineering
CNS	114	Center for Nanoscale Systems at Harvard University
d	6	Landau level degeneracy per unit area; $d \equiv N/L_x L_y = eB/h = B/\Phi_0$
$\mathcal{D}$	20	an operator used in the Jain hierarchy; represents formation of composite fermion Landau levels
$\mathbf{da}$	15	normal vector to an infinitesimal area in a Faraday loop; used in the flux-threading argument
DC	41	direct current
DI	116	deionized (water)
$\mathbf{dl}$	15	infinitesimal boundary vector of a Faraday loop; used in the flux-threading argument
e	4	magnitude of the electron charge
e^*	28	quasiparticle charge
E	5	energy of a quantum state; may be indexed by appropriate quantum numbers
$\mathbf{E}$	3	electric field vector, with magnitude E and components E_x, E_y, and E_z

e-beam	114	electron beam; refers to e-beam lithography or e-beam evaporation
EML	115	Exploratory Materials Laboratory; a cleanroom that is part of MTL at MIT
f	10	number of filled Landau levels
$\mathbf{F}$	3	Lorentz force vector
$F(g,x)$	72	a function given by Equation (3.4)
FWHM	62	full-width at half maximum
g	28	quasiparticle interaction parameter from chiral Luttinger liquid theory
g^*	10	effective Landé g-factor; $g^* = 0.4$ in GaAs
g_t	29	differential tunneling conductance
h	6	Planck constant
$\hbar$	5	reduced Planck constant
H	5	quantum mechanical Hamiltonian; we use separation of variables to consider H_{xy}, which is function of only x, y, p_x, and p_y
i	8	unit imaginary number
I	4	current passing through the sample; may be indexed to refer to the current due to a single eigenstate
I_0	4	total current impinging on the QPC; $I_0 = I + I_R$
I_c	94	critical current, above which breakdown of the QHE occurs
I_R	32	current reflected at the QPC; we consider the case $I_R = I_t$
I_t	32	tunneling current at the QPC
IpA	115	Isopropyl Alcohol
j	5	quantum number indexing the Landau levels
$\mathbf{J}$	8	current density vector, with components J_x and J_y (for a two-dimensional sample); the radial component J_r is used for the flux-threading argument (page 15)
k, l	14, 26	sets of indices; used to index electrons in Laughlin's Hamiltonian and trial wave function and to index ohmic contacts in the Landauer-Büttiker formalism
$\mathbf{k}$	5	wave vector, with components k_x, k_y, k_z
k_B	72	Boltzmann constant
k_F	49	Fermi wave vector; we take $k_F = \sqrt{2\pi n_{QPC}}$ in the QPC
l_0	6	magnetic length; $l_0 = \sqrt{\hbar/eB}$
l_c	49	classical cyclotron radius; $l_c = \hbar k_F/eB$
$\mathcal{L}$	20	an operator used in the Jain hierarchy; represents addition of a filled underlying Landau level

L_x, L_y, L_z	4	dimensions of the conducting sample (assume to be rectangular); for a two-dimensional sample, only L_x and L_y are relevant
m	6	an integer
m^*	5	effective mass of electrons in GaAs; $m^* = 0.067m_e$
m_e	5	mass of the electron
MBE	34	molecular beam epitaxy
MIT	xi	Massachusetts Institute of Technology
MTL	114	Microsystems Technology Laboratories at MIT
n	4	density of charge carriers (usually electrons) in the sample; for two-dimensional samples n is a sheet density
n_{QPC}	49	electron density in the QPC
N	6	number of states per Landau level; $N \equiv L_y/\Delta y_0 = eBL_xL_y/h = L_xL_y/l_0^2$
N_B	26	number of edge states in the sample far away from the QPC
N_D	29	number of edge states transmitted through the QPC
N_k^{out}, N_l^{in}	26	number of edge states outgoing from or incoming to an ohmic contact; indexed by contact k or l
N_{QPC}	48	number of one-dimensional states passing through a QPC modeled as an infinitely long potential well
p	20	an integer, used to write fractional filling factors
$\mathbf{p}$	5	momentum vector, with components p_x, p_y, and p_z
P	89	power
q	3	electric charge of a charge carrier
q	14	a positive odd integer, used to write fractional filling factors
Q	15	total charge that accumulates in a fractional quantum Hall system due to threading a flux quantum
QHE	1	quantum Hall effect
QPC	2	quantum point contact
QUILLS	88	quasi-elastic inter-Landau-level scattering
r	15	radial coordinate in a Corbino disk; used in the flux-threading argument
r_1, r_2	15	inner and outer radii of a Corbino disk; used in the flux threading argument
r_∞	63	high DC bias background of r_D at $\nu = 5/2$; accounted for with a fit parameter when fitting quasiparticle tunneling conductance
R, R_{XX}	11	longitudinal resistance (see also page 42 for a diagram); r_{XX} is the differential longitudinal resistance

R_D^+	32	diagonal resistance across the QPC (see also page 42 for a diagram); r_D^+ is the differential diagonal resistance and is equal to r_D in the quantum Hall regime
R_H, R_{XY}	4	Hall resistance (see also page 42 for a diagram); r_H or r_{XY} is the differential Hall resistance
RMS	41	root-mean-squared; for a sinusoidal wave is equal to the amplitude divided by $\sqrt{2}$
RPM	116	rotations per minute
SEBL	114	Scanning-Electron-Beam Lithography facility at MIT
t	15	time; used as a variable of integration in the flux-threading argument
T	59	electron temperature
T_B	82	temperature scale reflecting the strength of edge channel interaction in an expansion of exact quasiparticle tunneling conductance
TCE	115	Tricholorethylene
UV	117	ultraviolet
$\mathbf{v}$	3	drift velocity vector of a charge carrier, with magnitude v
V_D^+	32	diagonal voltage (corresponding to R_D^+); equal to V_H when there is no tunneling at the QPC or to zeroth order for weak tunneling
V_G	44	voltage applied to the specified gates (other gates may be grounded)
V_G^{anneal}	43	voltage at which the specified gates are annealed (other gates may be grounded); after annealing the gate voltage is changed from this value
V_G^{meas}	60	the gate voltage at which a particular measurement is performed
V_H	4	Hall voltage
V_t	32	tunneling voltage at the QPC
$V(z)$	4	electron confinement potential in the z direction; provided by the GaAs quantum well
W	49	estimated width of the QPC
x	34	fraction of Ga atoms replaced by Al in AlGaAs
x, y, z	3	the coordinates of three-dimensional space; unit vectors are given by $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\hat{\mathbf{z}}$
y_0	5	an offset to the guiding center of a Landau level plane wave in the Landau gauge; $y_0 = \hbar k_x/eB$; the spacing between guiding centers is $\Delta y_0 = 2\pi\hbar/eBL_x$

y_1	7	an offset to the guiding center of a Landau level plane wave in the Landau gauge due to a transverse electric field; $y_1 = -eE/m^*\omega_c^2$
Y	8	variable of integration, used to calculate the current carried by Landau level eigenstates; $Y = y - y_0 - y_1$
z	14	complex coordinates; $z = x + iy$; used in Laughlin's Hamiltonian and trial wave function

Appendix B

Device Fabrication Procedures

The measurements in this thesis were performed on a chip fabricated by Jeff Miller, using facilities at the Harvard Center for Nanoscale Systems (CNS) cleanroom. The specific fabrication procedures he used are described in his thesis [121]. We have adapted the fabrication procedures for use at the MIT Microsystems Technology Laboratories (MTL) cleanroom and Scanning-Electron-Beam Lithography (SEBL) facility (although some tools at Harvard are still used). These procedures are described in Section B.1. Section B.2 contains a detailed procedure for electron beam (e-beam) lithography using the MIT SEBL Raith tool, which is adapted from the procedure included in Jeff's thesis [121].

B.1 Fabrication Procedures

These fabrication procedures are written for future use by the Kastner group at MIT, and make use of facilities at Harvard and MIT. Facilities at Harvard are run

by CNS. At MIT we mainly make use of the MTL Exploratory Materials Laboratory (EML) cleanroom and the SEBL Raith tool. Other tools run by the Center for Materials Science and Engineering (CMSE) or individual labs are also sometimes used. Other users should adapt the specifics of these procedures to conform to the requirements of available tools and processes.

1. Cleave the chip. Aim for 5×5 mm in size; larger chips will not fit in the chip carrier, while for smaller chips the edge bead (increased thickness of resist near the chip edge) will infringe on the pattern. One can use either a hand-held diamond tip scriber (recommended) or the Loomis LSD-100 scriber/cleaver at Harvard. If using the LSD-100 a tool pressure of 8 kPa is reasonable, but be careful not to chip the surface when aligning or to contaminate the surface while breaking the chip.

2. Clean the chip by sonicating for 5 minutes each in Trichloroethylene (TCE), Acetone, and Isopropyl Alcohol (IpA) consecutively. Blow dry with compressed Nitrogen immediately upon removing from the IpA. To do so, hold the chip with tweezers on a clean wipe and blow perpendicularly onto the surface with a compressed Nitrogen gun. Holding the tweezers parallel to the plane of the chip allows one to bring the Nitrogen gun close to the surface of the chip in order to better remove the IpA. If small spots of dried liquid can be seen on the surface under a microscope afterwards, one should blow dry for a longer time. The TCE removes organic material from the chip and should not be necessary in later cleaning steps.

3. Spin OCG 825 photoresist onto the chip. The chip should be spun at $\sim$ 500–

700 RPM while dispensing the OCG (taking up to $\sim$ 10 s) and then quickly increased to 4000 RPM for 45 s. After spinning, prebake the chip in the EML 90 °C convection oven. When baking, place the chip in a aluminium tray, but do not place this tray directly on top of the wire rack; doing so could lead to uneven heating from the rack. Rather, place the tray on top of another, upside down tray.

4. Expose the edge bead and mesas using a photolithography mask. If using the Karl Suss MA 4 tool in EML, expose the edge bead first for 30 s and develop for 1 minute in OCG 934 1:1. Rinse in deionized (DI) water for 1 minute and blow dry after developing. Return to the MA 4, expose the mesas for 7.5 s, and repeat the develop and rinse steps. When using the MA 4, reduce the pressure to 3 bar and the wedge error to 0.3 bar so as to avoid cracking the chip when contacting the mask. We do not perform a postbake step because it can make the OCG very difficult to remove after etching the mesas.

5. Etch the mesas in 240:8:1 H_2O:H_2O_2:H_2SO_4 and rinse in DI water for 30 s. This mixture produces an etch rate of usually slightly less than 2 nm/s. However, this rate does vary so it is important to perform the etch in multiple steps and check the total etched depth after each step. If working with a real GaAs heterostructure one needs a test chip (cleaved from a pure GaAs wafer) also with the mesa pattern exposed in OCG to etch in parallel. Before beginning to etch, measure the height of OCG on the test chip with the Dektak profilometer in EML (do not risk damaging the real heterostructure with the profilometer). If this tool isn't working, the CMSE P16 Dektak profilometer can be used. Etch

and rinse both chips consecutively, aiming for perhaps $\sim 70\%$ of the desired depth. Then measure using the profilometer again to find the actual depth that was etched. Use this to calculate the actual etch rate and iterate until reaching the desired depth. One should etch past the 2DES, or past the second donor layer in a chip with double-sided doping, plus $\sim 10\%$ as an error margin.

6. Clean the chip as above and spin PMMA e-beam resist. Use the same spinning technique and times as for OCG. Bake the chip on a hot plate at 180 °C for 10 minutes. Ensure that the chip is in good thermal contact with the hot plate. One may wish to create a multi-layer stack of PMMA in order to create undercut or to increase the thickness. To do so, bake 5 minutes between each layer and again 10 minutes at the end. For ohmics, first spin a layer of PMMA 495 A11 and then a layer of 950 A4 to produce a thick layer with an undercut. If the ohmic stack is especially thick, a second layer of A11 may be added before the A4.

7. Expose PMMA using the SEBL Raith to form the pattern for ohmic contacts. Develop for 90 s in a mixture of 3:1 IpA:MIBK (sold pre-mixed) and rinse in IpA for 15 s. One should have previously performed a dose test to find the correct dose; generally around $700\,\mu\mathrm{A\,s/cm^2}$ is a good dose for ohmics. The specific procedure for using the Raith is included in Section B.2. Also expose bonding pads for the gates in this step.

8. Perform a UV/ozone clean using the Samco UV-1 at Harvard to remove any bits of PMMA left after developing. Clean for 2 minutes at 60 °C with a flow

rate of 1.0 L/min. Do not use a plasma clean because it has been observed to reduce the mobility of the 2DES.

9. Evaporate the ohmics using electron beam (e-beam) evaporation. Usually the EML Sloan e-beam evaporator is used, but there are a number of other evaporators available at MIT and Harvard. The exact ohmic recipe should be tailed to the specific heterostructure; a good starting point for high mobility material can be found in Jeff's thesis [121]. In general, evaporate alternating layers of Pt, Au, and Ge with thickness scaled based on the depth of the 2DES. Other recipes replace Pt with Ni. Immediately before evaporation a 3 s dip in NH_4OH with a 15 s DI water rinse should be performed to remove oxide from the surface. After evaporation, liftoff is performed by letting the chip sit in Acetone until the PMMA has dissolved, removing the metal not stuck to the surface of the chip. Some sonication may be necessary, but do so only in short (~ 5 s) bursts in order to avoid removing the actual ohmic stack. It is often useful to check the chip under a microscope after each burst to determine if the liftoff has finished correctly; to do so, use a small plastic chip carrier filled with Acetone to hold the chip while examining it. Do not let the chip dry prematurely. Once the liftoff has been deemed satisfactory, rinse in IpA and blow dry with Nitrogen.

10. Anneal the ohmics in the CNS RTP-600xp. Again, the proper recipe must be developed for each heterostructure. For high mobility material 100 s at 530 °C seems to be a good starting point. A sample recipe called 'Colin200nmedit' can be found on the RTP-600xp. In general, start by purging with 20,000 sccm of N_2 and then heating the chip to ~ 120 °C for roughly 1 minute to evaporate any

moisture. Ramp to the annealing temperature slowly enough that there is no significant overshoot in temperature. While heating, flow a N_2/H_2 forming gas of 1000 sccm. After annealing, let the sample cool and purge the system before removing.

11. Clean the chip and spin PMMA as above, this time for use in the small gate layer. In this case, spin only a single layer of PMMA 950 A4.

12. Expose the small gate layer using the Raith. As with the ohmics, the proper dose needs to be determined by testing; 800 μA s/cm^2 is a good starting point, though. Develop for 60 s and rinse in IpA for 15 s. Do not repeat the UV/ozone clean as it hinders proper liftoff of the small gates.

13. Optional: Etch under the small gates using a reactive ion dry etch (CNS Unaxis Shuttleline ICP) followed by a 495:4:1 H_2O:H_2O_2:H_3PO_4 wet etch. These etch steps have only recently been added to the fabrication procedure and the resulting chips are still being tested (although the Marcus group at Harvard has used a similar recipe with success [122]). The dry etch uses BCl_3 (15), Ar (7.5), and N_2 (3.8), with flow rates in sccm in parentheses, and a total pressure of $\sim$ 3.0 mTorr (or as low as the system will go). The ICP RF power is 500 W and the Bias RF is 135 W, although reducing to 100 W is recommended to decrease the etch rate and produce more consistent results. The gas flow is stabilized for 30 s before applying the power and the chuck is kept at 20 °C. Etch times should be tested to achieve the desired depth; as a reference, 17 s produces $\sim$ 80 nm of etch depth. The depth of the dry etch should be past the donor layer but not

close to the 2DES; if a GaAs test chip is etched simultaneously an atomic force microscopy (AFM) measurement of the depth can be performed on it using the CMSE AFM or the one owned by the Bulovic group. After the dry etch, a shallow wet etch of $\sim 20\,\text{nm}$ is performed. The advantage of using the etch mixture listed above is that it etches GaAs and AlGaAs at equal rates [123] (roughly $0.4\,\text{nm/s}$), allowing the etch depth of a GaAs test chip to be measured in place of a chip with real heterostructure. After the wet etch another AFM measurement may be performed to find the total depth of the trench.

14. Evaporate 5 nm of Cr and 15 nm of Au with an e-beam evaporator and perform liftoff to form the small gates. Do not perform the Na_4OH dip as done for the ohmics. If the etch procedure above has been performed then the Cr thickness should be increased to $\sim 10\,\text{nm}$ and the Au thickness increased so the total gate thickness is $\sim 80\%$ of the total etch depth.

15. Clean the chip and spin PMMA as above, for the connector gate layer. These gates connect the small gates that form the device to the bonding pads and must pass across the edge of the mesa. Hence, they should be evaporated thick enough to not break, requiring a thicker layer of PMMA than for the small gates. Using one layer of 495 A11 and then one layer of 950 A4 is usually enough and also provides an undercut to help liftoff.

16. Expose the connector gate layer using the Raith. As with the ohmics, good doses are usually around $700\,\mu\text{A s/cm}^2$. Develop for 90 s and rinse in IpA for 15 s. We have found that removing the bonds to the gate bonding pads (for

example if the bond doesn't stick the first time) causes the metal deposited in the next step to come off the chip. To solve this problem, only overlap the connector gates with the bonding pads (which were formed with the ohmics) by about 1/3 the size of the bonding pad (which are usually $\sim (150\,\mu\text{m})^2$).

17. Perform the same UV/ozone clean as for the ohmics (but not the Na_4OH dip) and evaporate metal for the connector gates. The total thickness should be the depth of the mesa etch plus $\sim 10\%$ and should be comprised of roughly 5% Cr as an adhesion layer and the rest Au. Liftoff as described above.

18. The chip is ready to be attached and wire bonded to the chip holder. Use a toothpick or something similar to place a small drop of PMMA in the middle of the chip carrier (if possible, use old PMMA that is no longer being used as a resist). With tweezers, carefully place the chip in the holder, on top of the PMMA. Let the PMMA dry and it will glue the chip in place. Use either the CNS or the Jarillo-Herrero wire bonder to bond the bonding pads on the chip to those on the chip carrier. These two wire bonders are both manufactured by West Bond and are very similar. Setting the power to 300 and the bond time to 20 ms has proven to generally produce reliable bonds.

B.2 Electron Beam Lithography Procedure

These procedures for using the Raith e-beam lithography tool at MIT have been adapted from those in Jeff's thesis (written for the Harvard Raith) [121]. They are written for version 4 of the Raith software, but version 5 seems similar (and the MIT

Raith will likely continue to use version 4).

1. The chuck is not the standard chuck used by other Raith users. Instead use the chuck with built-in clips along two sides. Before anything else, blow the top and bottom of the chuck (especially the indentations on the bottom) with the Nitrogen gun to remove any dust. Secure the chip to the chuck using one of these clips. Take care to ensure that the clip does not extend too far onto the chip where it might cover part of the device. After clamping it down, gently press against the edge of the chip using tweezers to ensure that it is flat against the chuck. If the chip rocks back and forth upon applying pressure it isn't completely flat and charging effects may cause problems; reposition the chip slightly and try again.

2. Use the precision pipette to put a small drop of solution with Gold nanocrystals at each corner of the chip. Start with a smaller volume setting on the pipette (50 nL is usually reasonable) and only increase it if needed; even a very small droplet will work fine. Use the solution with 100 nm diameter nanocrystals. Let the solution completely dry (the droplet will visibly disappear) before putting the chuck in the Raith.

3. Blow the bottom of the chuck with Nitrogen again and load it into the Raith, following standard procedure.

4. Once the chuck has been loaded, set the acceleration voltage to 10 kV and the aperture to 30μm. The working distance depends on the thickness of the chip, but around 6.4 mm is a good starting value. Quickly and repeated press the two

black buttons on the piezo controller to shake the stage into a stable position. If the controller overloads turn it off and back on and try again. Raise the stage to $Z = 18.8\,\text{mm}$; this is different from the normal value because the chuck has a different thickness.

5. Using TV mode, move the gun to above the clip holding the chip and zoom out. Switch to InLens mode and move down the clip to find the boundary of the chip. Then move along the boundary of the chip to a corner and focus on the gold nanocrystals. Make sure to only view the chip near the edge, so as to not expose around the mesas. Try to focus on individual or small groups of nanocrystals found outside the evaporation ring, so that they are closer to the surface of the PMMA rather than clumped up. It is helpful to record the X and Y position as well as the focusing parameters (working distance, stigmation, and aperture alignment) at this point. Repeat for two other corners, and record the XY location of the gold.

6. Move from one of the corners towards the middle of the chip to find the alignment marks (for the ohmic layer these are mesa features, although for small gates and connector gates one can align to metallic alignment marks written during the previous Raith exposure). Use these alignment marks to set the UV origin and angle using the 'Adjust UVW' window. It is easier if focus correction is not enabled at this point. Alternatively if this is a dose test on a chip without mesas, use the corners of the chip to define the UV-coordinate system.

7. At this point, set the acceleration voltage and aperture to the values to be

used for the actual exposure. For ohmics and connector gates use 30 kV and 120 μm. For small gates use 30 kV and 30 μm. At 30 kV use the SE2 detector. Unfortunately most features are less clearly visible with these settings—this is why we find the gold nanocrystals and define the UV-coordinate system first at 10 kV. Before continuing, return to the gold nanocrystals and focus again.

8. Measure the current using the Faraday cup, which is located near the intersection of the two bars holding the clips along the sides of the chuck. One should measure the current at a consistent magnification each time; an easy way to do this is to zoom to the highest magnification before measuring. Reasonable values are a few nA for ohmics and large gates and a few hundred pA for small gates. After measuring the current, set the writefield to 100 μm using the 'Microscope Control' window, which also sets the current magnification to 620×.

9. Return to one of the gold nanocrystals (most easily by moving to the XY values recorded earlier) and take an image using the Raith software. Change the center of the image by using Ctrl-right click to indicate the new center. Zoom in by right clicking and choosing 'Select Area'. Hold Ctrl while drawing the zoom box to preserve the aspect ratio and then right click and select 'Zoom In'. Iterate these steps a few times to zoom in to maximum magnification and center the image on a well-defined feature, such as a nanocrystal. Once the small crosshairs (indicating the center of the image) are on a well-defined feature, open a 'New positionlist' and drag the 'Coarse', 'Medium', and 'Fine' manual writefield alignment procedures onto the positionlist from the 'Scan

Manager' window. Run each of these in turn, making sure each produces a reasonable alignment before moving on to the next.

10. Define the UV-coordinate system again, using the origin and angle procedures as above. Double check alignment at other alignment marks on the sample. Now use the three point alignment with focus correction (remember to enable focus correction) at gold nanocrystals in three of the corners. However, this step is only to provide focus correction for the sample, and not to adjust the UV-coordinate system. Hence, copy the current UV values for each of the three points when doing the alignment. Unfortunately, the first three point alignment never works properly, so repeat the procedure at the three points. Remember to uncheck the three check boxes in the 'Adjust UVW' window before refocusing. In addition, it important to not move the stage during this time, as doing so will change the UV-coordinates and invalidate the alignment. If once can get the 'Spot Exposure' function to work, it is likely preferable to use that to focus (as described in Jeff's thesis [121]) during this step, rather than using the gold nanocrystals. This would also allow one to focus much closer to the region to be exposed.

11. Repeat the writefield alignment on the gold nanocrystals using the 'Fine' manual alignment. Repeat the procedure until the correction at each step is minimal (usually twice is enough). Check the writefield parameters using the 'Align Writefield' window or the Internet Explorer link. Zoom values should be around $0.93\times$, shift values roughly $0.2\,\mu$m or less, and rotation values a few degrees or less. Check the UV alignment at various alignment marks a final time.

Alignment needs to be good to at least 5 μm, which is roughly the overlap between the small and connector gates. However, alignment within 1 μm is usually possible.

12. Set the area dose to 100.0 μA s/cm^2. Set the area step size to 0.2 μm for ohmics and connector gates and to 0.03 μm for small gates. Use the calculator button next to 'Area Dwell Time' to calculate the dwell time based on the dose and step size. The beam speed will read a fairly large value at this point (tens of mm/s) but in actuality will be lower once the dose factor is set. Create a 'New positionlist' and drag the loaded design file onto it from the 'GDSII Database' window. Set the 'Dose Factor' for the overall pattern in the 'Exposure Properties' window (right click on the file in the positionlist and choose 'Properties'). The actual applied dose is the area dose times this dose factor. Alternatively, one can edit the design file (which must be saved as a `.csf` file) and modify the dose factor for individual elements by using the editing features of the Raith software (choose 'Modify: Dose Factor: Scale'). Also in the 'Exposure Properties' window, set the layers to be exposed and the *UV* 'Position' (corresponding to the center of the bottom left writefield). All the other parameters should be correct by default, but it doesn't hurt to double-check. The pattern is ready for exposure.

13. While the pattern is being exposed, check that the beam sweeper and beam blanker all are working correctly and that the current reading increases when the beam is unblanked. It can also be useful to keep an eye on the exposure progress by viewing the pattern (click 'View' in the 'GDSII Database' window;

the crosshairs indicate the current writefield position). After the exposure is finished check the current again at the Faraday cup to make sure it hasn't changed significantly. Unload the chip.

Appendix C

Code for Data Collection and Analysis

In this appendix we present computer code used for data collection and analysis. The code is written for IGOR Pro, which is sold by Wavemetrics (`www.wavemetrics.com`). It was developed and run using version 6.1.2.1 of IGOR; however, future versions of IGOR 6 *should* be backwards compatible. The NIDAQ Tools MX IGOR Package is also required in order to interface with the National Instruments PCI-GPIB board (using NI-DAQmx 8.9 and the NI-288.2 driver, version 2.7.1). This code was written primarily by Colin Dillard with contributions from Xi Lin (especially for the `Constants.ipf` and `DataSearching.ipf` procedure files). The basic algorithms used for data collection (used in `DataCollection.ipf`) were inspired by code previously used by Iuliana Radu, Jeff Miller, Dominik Zumbühl, and Alex Johnson. However, the actual IGOR procedures have been heavily modified to add functionality and to improve error handling, data retention, and clarity. Similarly, code to

interface with various electronics (`A33220A.ipf`, `DMM.ipf`, `Initializations.ipf`, `SMSMagnetPowerSupply.ipf`, and `Yokogawa.ipf`) has been updated to improve reliability and clarity.

The general approach used to collect and store data is described in Section C.1. The IGOR code is contained in IGOR procedure files (extension `.ipf`), each of which contains a number of related procedures. Each such procedure file is contained in its own section, as follows:

- Code to interface with electronics

  Section C.2: create a new experiment and initialize all electronics

  Section C.3: control the Agilent voltage source used to apply AC and DC bias

  Section C.4: control the Yokogawa voltage sources used to apply voltage to the gates

  Section C.5: control the magnet power supply

  Section C.6: read output from the digital multimeters

- Code to collect, handle, and analyze data

  Section C.7: control experimental parameters and collect data

  Section C.8: save and transfer data between computers

  Section C.9: search through data based on any parameter

  Section C.10: physical constants

  Section C.11: common procedures used for data analysis

Section C.12: fitting functions beyond those included in IGOR

Unfortunately, IGOR has no character for line continuation and this has led to a number of lines of code extending beyond the margins of this thesis. The offending code, which is mostly either comments (preceded by `//`) or long error messages, has been rearranged into multiple, shorter lines. The code may thus be a bit more awkward in some places, but no functionality should be impaired.

C.1 General Approach to Data Collection and Storage

The basic data structure in IGOR is called a wave, which can be thought of as a matrix of values (or strings) with defined dimensions. The units and scaling of each dimension can be defined. We use primarily one-dimensional waves to store data, although two-dimensional waves are used for measurements in which two independent variables are controlled. To collect a one-dimensional wave the user inputs the number of points to be taken as well as the starting and ending values of the independent variable and its rate of change. The code divides the range of the independent variable into a corresponding number of discrete steps and progressively increments along these steps at the desired rate. At each step the desired dependent variables are measured from digital multimeters via GPIB connections and the values are stored in separate waves. The waves are saved with a naming system that records the dependent and independent variables along with a global ID number.

The specifics of sweeping each independent variable depends on the type of equip-

ment used and is handled by the corresponding procedures. For example, gate voltages are applied with Yokogawa voltage sources and the function `SetYoko` is the one used to actually send the GPIB command to change the output voltage. In this manner the experimenter can use same function to collect all one-dimensional waves and merely needs to specify the correct independent variable for each; the code handles all the details of sweeping the independent variable. The one exception to this elegant method is magnetic field. Unfortunately the magnet power supply produces uncorrectable errors when attempting to repeatedly step it in the manner described above. Hence, one must command the power supply to perform a continuous field sweep and separately measure at discrete points in time during the sweep. Fortunately, the value of the magnetic field can be monitored via the reported current output of the power supply and, hence, the collected data can be plotted against the measured magnetic field to produce the desired wave.

One guiding principal in writing this code was that all experimental conditions and all measured data should be saved (and accessible to later search). Instead of searching page by page through old log books for something that may not even have been recorded one can browse and search a comprehensive digital record. We have not quite achieved this ideal (for example, gate annealing times are not automatically recorded), but the current system has proven quite useful. In order to implement this system two instances of IGOR are run simultaneously for each experiment (use Ctrl-double click to open a second instance). The "Data Collection" (or"Run") instance is used only to control the experiment and collect data. It saves all data and experimental conditions in the experimental folder as well as a back-up on a separate

hard drive. The "Evaluation" instance is used to view data after collection and for preliminary data analysis. In this manner one can work with recently collected data while continuing to collect data using the data collection instance. Waves can also be saved (usually in the form of a graph) and transferred to another computer for more in-depth data analysis. Search functionality is provided to find specific types of measurements that were performed. In a similar spirit, we have found it useful to implement these procedure files as 'Global' procedures, shared by all experiments. This eliminates the need to keep track of different versions of the procedure files (and which experiment files they belong to) and automatically propagates any changes (such as bug fixes) to all experiment files. One just needs to make sure to maintain the same version of the Global procedures on all computers used to collect or analyze data.

One major improvement in the length and readability of the code would be to condense the multitude of error messages. The error checking is quite comprehensive (mostly checking for bad or missing values of experimental parameters) but has proven to be highly repetitive among different procedures, leading to a large amount of duplicate or nearly-duplicate code. Collapsing the error checking into a dedicated procedure has the potential to substantially decrease the length and, hence, improve the readability of the code. However, if one is only concerned with the functionality of the code, no changes are required.

C.2 Initializations

There are two important functions in this procedure file. `NewExperiment` is run from a template IGOR experiment file to create a new experiment. The template file must include text waves `InVarWaveTemplate` and `DeVarWaveTemplate` containing lists of the independent and dependent variables, respectively, that are to be controlled or measured. Also, a text wave `ExperimentalDetailsTemplate` contains a list of all experimental parameters to be recorded while taking data. The other important function, `InitGPIB`, is used to set the GPIB address for the various pieces of electronic equipment.

```
#pragma rtGlobals=1     // Use modern global access method.

function NewExperiment(expname)
   string expname //name of experiment to create

   NewPath/C/O datapath "D:Data:"
   GetFileFolderInfo/Q/Z/P=datapath ("ExperimentNames.itx")
   if(V_flag == 0)
      LoadWave/Q/T/O/P=datapath "ExperimentNames.itx"
   else
      Print "ExperimentNames.itx not found in D:Data."
      Print "Unable to check for conflicting experiment names.  Aborting. (NewExperiment)"
      return -1
   endif
   Wave/T ExperimentNames
   string namesearch = TWave2Str(ExperimentNames, separator = ";")
   string searchstr = ""
   sprintf searchstr, "(?i)(;%s;)|(^%s;)", expname, expname
   if(GrepString(namesearch, searchstr))
      printf "File name %s already used by an Igor experiment.  ", expname
      printf "Please choose a different name. (NewExperiment)\r"
      return -1
   endif
   InsertPoints Inf, 1, ExperimentNames
   ExperimentNames[Inf] = expname
   Save/T/O/P=datapath ExperimentNames as "ExperimentNames.itx"

   string filename = expname + ".pxp"
   string pathstring = "D:Data:" + expname
   NewPath/C/O savepath pathstring //normal save path
   NewPath/C/O graphpath pathstring + ":Graphs" //path for graph
   NewPath/C/O binarypath pathstring + ":Binaries" //path for wave binaries
   pathstring = "E:Data:" + expname
   NewPath/C/O backuppath pathstring //path for backup data

   //update to include any new independent and dependent variables
   wave/T InVarWaveTemplate, DeVarWaveTemplate
   duplicate/O InVarWaveTemplate InVarWave
```

```
   duplicate/O DeVarWaveTemplate DeVarWave

   variable/G LoadedIndex = 0 //start index for loading binaries

   //create ExperimentalDetails from ExperimentalDetailsTemplate
   wave/T ExperimentalDetailsTemplate
   duplicate/O ExperimentalDetailsTemplate ExperimentalDetails
   Duplicate/O/R=[][0] ExperimentalDetails, ExperimentalDetailsName
   edit/K=2/W=(748.5, 55.25, 1074, 621.5)/N=DetailsTable ExperimentalDetails

   SaveExperiment/C/P=savepath as filename //save experiment as; 'evaluation' experiment
   filename = expname + "Run.pxp"

   datatypes() //open data collection panel

   NewNotebook/F=0/K=2/N=Record as "Record"
   MoveWindow/W=Record 437,0,875, 584
   string recordstr = date() + ": " + time() + ": " + "NewExperiment(" + expname + ")\r"
   Notebook Record, selection={StartofFile, StartofFile}, text = recordstr
   SaveNotebook/O/P=savepath/S=6 Record as "Record.txt"

   MoveWindow/W=DetailsTable 748.5, 139, 1074, 621.5
   DoWindow/F DetailsTable

   //global variables related to SMS
   variable/G SMSHeaterOnTime = NaN, SMSDirectionFlag = NaN
   //0: make new graphs for each data set, 1: add new data to current graphs
   variable/G KeepGraphsFlag = 0
   SaveExperiment/P=savepath as filename //save experiment as; 'data collection' experiment
end

function InitGPIB()
   // Store the GPIB indentifiers of the instruments in the corresponding variables
   variable/G gpibboard
   variable/G yoko0
   variable/G yoko1
   variable/G yoko2
   variable/G yoko3
   variable/G dmm1
   variable/G dmm2
   variable/G dmm3
   variable/G dmm4
   variable/G dmm5
   variable/G SMS
   variable/G A33220A

   // Get the GPIB identifiers
   execute "NI488 ibfind \"gpib0\", gpibboard"
   execute "NI488 ibfind \"dev11\", yoko0"
   execute "NI488 ibfind \"dev1\", yoko1"
   execute "NI488 ibfind \"dev3\", yoko2"
   execute "NI488 ibfind \"dev5\", yoko3"
   execute "NI488 ibfind \"dev10\", dmm1"
   execute "NI488 ibfind \"dev6\", dmm2"
   execute "NI488 ibfind \"dev7\", dmm3"
   execute "NI488 ibfind \"dev9\", dmm4"
   execute "NI488 ibfind \"dev8\", dmm5"
   execute "NI488 ibfind \"dev4\", SMS"
   execute "NI488 ibfind \"dev2\", A33220A"

   variable/G yoko4
   execute "NI488 ibfind \"dev12\", yoko4"
```

```
   // Make sure we're talking to the right board
   execute "GPIB board gpibboard"

   if(WaveExists(datatypedmm))
      updateDMM()
   endif
End

function updateDMM()
   wave/T datatypedmm
   variable numdmms = dimsize(datatypedmm, 0)
   variable n = 0
   string dmmname = ""

   for(n=0; n<numdmms; n+=1)
      dmmname = "dmm" + datatypedmm[n][2]
      NVAR dmm=$dmmname
      if(NVAR_exists(dmm)) //if dmm found update datatypedmm
         datatypedmm[n][3] = num2istr(dmm)
      else
         datatypedmm[n][3] = "0"
      endif
   endfor
end
```

C.3 Agilent Voltage Source

This is a set of procedures for interfacing with the Agilent 33220A AC and DC voltage source via GPIB. In the experiments reported in this thesis, this voltage source is changed to a current source using a large resistor (usually 200 MΩ) in series with the sample. The Agilent is initialized using `InitA33220A` and other functions are provided to turn the voltage source on and off, set AC and DC magnitudes, and to set the AC frequency.

```
#pragma rtGlobals=1     // Use modern global access method

function InitA33220A() //run every time after turning 33220A on
   //set correct values for "33220A Frequency", "SampleIac", and "VtoI R"
   //in ExperimentalDetails before running
   NVAR A33220A //GPIB address
   string commandstr
   WAVE/T ExperimentalDetails
   Duplicate/O/R=[][0] ExperimentalDetails, ExperimentalDetailsName
   variable VtoIR //resistor value used for voltage to current bias
   variable VdivAC //AC voltage divider value (>=1) for voltage bias
   variable VdivDC //DC voltage divider value (>=1) for voltage bias
   sprintf commandstr, "GPIB device %f", A33220A //choose GPIB device
   execute commandstr
```

```
variable outputstatus = ReadA33220AOutput() //read if output is on or off
if(outputstatus == 1) //output on
   Print "33220A output is ON.  Not changing parameters.  Please turn output off to change."
elseif(outputstatus == 0) //output off
   Print "33220A output is OFF."

   execute "GPIBwrite/F=\"FUNC SIN\\n\"" //output as sine wave
   Print "33220A set to Sine waveform."
   execute "GPIBwrite/F=\"Voltage:unit VRMS\\n\"" //AC specified in RMS
   Print "33220A set to RMS voltage."

   FindValue/TEXT=("33220A Freq")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      //get frequency value from ExperimentalDetails
      variable freq = str2num(ExperimentalDetails[V_value][1])
      if(!StringMatch(num2str(freq), "NaN"))
         SetA33220AFreq(freq) //set 33220A frequency to freq
         printf "33220A Frequency set to %g Hz.\r", freq
      else
         Print "Recorded \"33220A Freq\" in ExperimentalDetails not valid."
         Print "Cannot set frequency. (InitA33220A)"
      endif
   else
      Print "\"33220A Freq\" not found in ExperimentalDetails."
      Print "Cannot set frequency. (InitA33220A)"
   endif

   FindValue/TEXT=("VacDivider")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      VdivAC = str2num(ExperimentalDetails[V_value][1]) //voltage to current bias resistor value
      if(VdivAC < 1)
         printf "Warning: recorded \"VacDivider\" value %g nonphysical; must be >= 1. ", VdivAC
         printf "Voltage bias will not work. (InitA33220A)\r"
      elseif(StringMatch(num2str(VtoIR), "NaN"))
         Print "Recorded \"VacDivider\" in ExperimentalDetails not valid."
         Print "Voltage bias will not work. (InitA33220A)"
      else
         printf "VacDivider = %g.\r", VdivAC
      endif
   else
      Print "Warning: \"VacDivider\" not found in ExperimentalDetails."
      Print "Voltage bias will not work. (InitA33220A)"
   endif
   FindValue/TEXT=("VdcDivider")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      VdivDC = str2num(ExperimentalDetails[V_value][1]) //voltage to current bias resistor value
      if(VdivDC < 1)
         printf "Warning: recorded \"VdcDivider\" value %g nonphysical; must be >= 1.  ", VdivDC
         printf "Voltage bias will not work. (InitA33220A)\r"
      elseif(StringMatch(num2str(VtoIR), "NaN"))
         Print "Recorded \"VdcDivider\" in ExperimentalDetails not valid."
         Print "Voltage bias will not work. (InitA33220A)"
      else
         printf "VdcDivider = %g.\r", VdivDC
      endif
   else
      Print "Warning: \"VdcDivider\" not found in ExperimentalDetails."
      Print "Voltage bias will not work. (InitA33220A)"
   endif

   FindValue/TEXT=("VtoI R")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
```

```
         VtoIR = str2num(ExperimentalDetails[V_value][1]) //voltage to current bias resistor value
         if(!StringMatch(num2str(VtoIR), "NaN"))
            printf "VtoI R = %g Ohms.\r", VtoIR
            FindValue/TEXT=("SampleIac")/TXOP=4 ExperimentalDetailsName
            if(V_Value > -1)
               variable ACnA = str2num(ExperimentalDetails[V_value][1]) //AC amplitude
               if(!StringMatch(num2str(ACnA), "NaN"))
                  SetA33220AAC(ACnA*1e-9*VtoIR) //set AC voltage output on 33220A
                  printf "33220A AC output set to %g V.\r", ACnA*1e-9*VtoIR
               else
                  Print "Recorded \"SampleIac\" in ExperimentalDetails not valid."
                  Print "Cannot set AC amplitude. (InitA33220A)"
               endif
            else
               Print "\"SampleIac\" not found in ExperimentalDetails."
               Print "Cannot set AC amplitude. (InitA33220A)"
            endif
         else
            Print "Recorded \"VtoI R\" in ExperimentalDetails not valid."
            Print "Cannot set AC amplitude. (InitA33220A)"
         endif
      else
         Print "\"VtoI R\" not found in ExperimentalDetails.  Cannot set AC amplitude. (InitA33220A)"
      endif

      SetA33220ADC(0) //set DC voltage output to 0 V on 33220A
      Print "33220A DC output set to 0 V."
   endif
end

function A33220AOn() //turn 33220A output on
   NVAR A33220A

   variable dcoutput = ReadA33220ADC()
   if(dcoutput != 0)
      printf "DC offset %g V is nonzero.  ", dcoutput
      printf "Please set to 0 V before turning on. (A33220AOn)\r"
      return -1
   endif

   string commandstr
   sprintf commandstr, "GPIB device %f", A33220A
   execute commandstr
   execute "GPIBwrite/F=\"OUTP ON\\n\""
end

function A33220AOff() //turn 33220A output off
   NVAR A33220A

   variable dcoutput = ReadA33220ADC()
   if(dcoutput != 0)
      printf "DC offset %g V is nonzero.  ", dcoutput
      printf "Please set to 0 V before turning off. (A33220AOn)\r"
      return -1
   endif

   string commandstr
   sprintf commandstr, "GPIB device %f", A33220A
   execute commandstr
   execute "GPIBwrite/F=\"OUTP OFF\\n\""
end

function ReadA33220AOutput() //check if 33220A output is on (return 1) or off (return 0)
```

```
   NVAR A33220A
   string commandstr
   string/G junkstring
   sprintf commandstr, "GPIB device %f", A33220A
   execute commandstr
   execute "GPIBwrite/F=\"OUTP?\\n\""
   execute "GPIBread/T=\"\\n\" junkstring"
   return str2num(junkstring) //0: output off, 1: output on
end

function SetA33220AFreq(frequency) //set 33220A frequency in Hz
   variable frequency //Frequency in Hz
   NVAR A33220A
   WAVE/T ExperimentalDetails
   Duplicate/O/R=[][0] ExperimentalDetails, ExperimentalDetailsName
   string commandstr, freqstr
   sprintf commandstr, "GPIB device %f", A33220A
   execute commandstr
   sprintf commandstr, "GPIBwrite/F=\"FREQ %f\\n\"" , frequency
   execute commandstr
   FindValue/TEXT=("33220A Freq")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      sprintf freqstr, "%g", frequency
      ExperimentalDetails[V_value][1] = freqstr //update "33220A Frequency" in ExperimentalDetails
   else
      Print "Warning: \"33220A Freq\" not found in ExperimentalDetails."
      Print "Unable to record frequency. (SetA33220AFreq)"
   endif
End

function SetIAC(Amplitude) //Set AC current amplitude
   variable Amplitude //in nA

   if((amplitude<0) || (amplitude>10)) //enforce 0 to 10 nA range
      printf "AC amplitude %g nA is out of range.  ", amplitude
      printf "Please use 0-10 nA or change program. (SetIAC)\r"
      return -1
   endif

   variable VToIR = 0
   WAVE/T ExperimentalDetails
   Duplicate/O/R=[][0] ExperimentalDetails, ExperimentalDetailsName

   FindValue/TEXT=("VtoI R")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      VtoIR = str2num(ExperimentalDetails[V_value][1]) //voltage to current bias resistor value
      if(!StringMatch(num2str(VtoIR), "NaN"))
         SetA33220AAC(Amplitude*1e-9*VtoIR) //set AC voltage output on 33220A
         printf "AC current at sample set to %g nA.\r", Amplitude
      else
         Print "Recorded \"VtoI R\" in ExperimentalDetails not valid."
         Pring "Cannot set AC current amplitude. (SetIAC)"
      endif
   else
      Print "\"VtoI R\" not found in ExperimentalDetails.  Cannot set AC current amplitude. (SetIAC)"
   endif
end

function SetVAC(Amplitude) //Set AC voltage amplitude
   variable Amplitude //in microV

   if(amplitude<0) //must be positive
      printf "AC amplitude %g microV is negative.  ", amplitude
```

```
      printf "Please use a positive voltage. (SetVAC)\r"
      return -1
   endif

   variable VDivAC = 0
   WAVE/T ExperimentalDetails
   Duplicate/O/R=[][0] ExperimentalDetails, ExperimentalDetailsName

   FindValue/TEXT=("VacDivider")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      VDivAC = str2num(ExperimentalDetails[V_value][1]) //voltage to current bias resistor value
      if(!StringMatch(num2str(VDivAC), "NaN"))
         SetA33220AAC(Amplitude*1e-6*VDivAC) //set AC voltage output on 33220A
         printf "AC voltage at sample set to %g microV.\r", Amplitude
      else
         Print "Recorded \"VacDivider\" in ExperimentalDetails not valid."
         Print "Cannot set AC voltage amplitude. (SetVAC)"
      endif
   else
      Print "\"VacDivider\" not found in ExperimentalDetails."
      Print "Cannot set AC voltage amplitude. (SetVAC)"
   endif
end

function SetA33220AAC(Amplitude) //Set the AC amplitude in volts
   variable Amplitude //Volts

   if((amplitude<0) || (amplitude>2)) //enforce 0 to 2 V range (10 nA with 200 MOhm resistor)
      printf "AC amplitude %g V is out of range.  ", amplitude
      printf "Please use 0-2 V or change program. (SetA33220AAC)\r"
      return -1
   endif

   NVAR A33220A
   WAVE/T ExperimentalDetails
   Duplicate/O/R=[][0] ExperimentalDetails, ExperimentalDetailsName
   string commandstr, ACnAstr, ACmicroVstr

   sprintf commandstr, "GPIB device %f", A33220A
   execute commandstr
   sprintf commandstr, "GPIBwrite/F=\"Voltage %f\\n\"", amplitude
   execute commandstr

   FindValue/TEXT=("VtoI R")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      //voltage to current bias resistor value
      variable VtoIR = str2num(ExperimentalDetails[V_value][1])
      if(!StringMatch(num2str(VtoIR),"NaN"))
         FindValue/TEXT=("SampleIac")/TXOP=4 ExperimentalDetailsName
         if(V_Value > -1)
            if(amplitude>0.0035) //3.5 mV AC RMS is lower limit of voltage generator
               sprintf ACnAstr, "%g", 1e9*amplitude/VtoIR
               ExperimentalDetails[V_value][1] = ACnAstr //update "SampleIac" in ExperimentalDetails
            else
               sprintf ACnAstr, "%g", 1e9*0.0035/VtoIR
               ExperimentalDetails[V_value][1] = ACnAstr
            endif
         else
            Print "\"SampleIac\" not found in ExperimentalDetails."
            Print "Unable to record AC current amplitude. (SetA33220AAC)"
         endif
      else
         Print "Recorded \"VtoI R\" in ExperimentalDetails not valid."
```

```
         Print "Unable to record AC current amplitude. (SetA33220AAC)"
      endif
   else
      Print "\"VtoI R\" not found in ExperimentalDetails."
      Print "Unable to record AC current amplitude. (SetA33220AAC)"
   endif

   FindValue/TEXT=("VacDivider")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      //voltage to current bias resistor value
      variable VDivAC = str2num(ExperimentalDetails[V_value][1])
      if(!StringMatch(num2str(VDivAC),"NaN"))
         FindValue/TEXT=("SampleVac")/TXOP=4 ExperimentalDetailsName
         if(V_Value > -1)
            if(amplitude>0.0035) //3.5 mV AC RMS is lower limit of voltage generator
               sprintf ACmicroVstr, "%g", 1e6*amplitude/VDivAC
               //update "SampleVac" in ExperimentalDetails
               ExperimentalDetails[V_value][1] = ACmicroVstr
            else
               sprintf ACmicroVstr, "%g", 1e6*0.0035/VDivAC
               ExperimentalDetails[V_value][1] = ACmicroVstr
            endif
         else
            Print "\"SampleVac\" not found in ExperimentalDetails."
            Print "Unable to record AC voltage amplitude. (SetA33220AAC)"
         endif
      else
         Print "Recorded \"VacDivider\" in ExperimentalDetails not valid."
         Print "Unable to record AC voltage amplitude. (SetA33220AAC)"
      endif
   else
      Print "\"VacDivider\" not found in ExperimentalDetails."
      Print "Unable to record AC voltage amplitude. (SetA33220AAC)"
   endif
end

function SetA33220ADC(Offset) //Set the DC offset in V
   variable Offset //Volts
   NVAR A33220A
   string commandstr

   sprintf commandstr, "GPIB device %f", A33220A
   execute commandstr
   sprintf commandstr, "GPIBwrite/F=\"Voltage:Offset %f\\n\"", Offset
   execute commandstr
end

function A33220AHighLoad() //Sets to High Impedance Mode
   NVAR A33220A
   string commandstr
   sprintf commandstr, "GPIB device %f", A33220A
   execute commandstr
   execute "GPIBwrite/F=\"Output:Load Max\\n\""
end

function SetA33220AAuto(value) //sets Auto Range feature of A33220A
   string value
   NVAR A33220A
   string commandstr, commandstr2

   //turn auto ranging on; will change range as needed; this produces a spike in the output
   if(StringMatch(value, "on"))
      commandstr2 = "GPIBwrite/F=\"VOLT:RANG:AUTO ON\\n\""
```

```
   //auto ranges to set voltage and does not change that range automatically
   elseif(StringMatch(value, "off"))
      commandstr2 = "GPIBwrite/F=\"VOLT:RANG:AUTO ONCE\\n\""
   else
      printf "Argument %s is not valid.  Use \"on\" or \"off\". (SetA33220AAuto)\r", value
      return -1
   endif

   sprintf commandstr, "GPIB device %f", A33220A
   execute commandstr
   execute commandstr2
end

Function ReadA33220ADC() //returns DC offset in Volts of 33220A
   NVAR A33220A
   string/G junkstring
   string commandstr
   sprintf commandstr, "GPIB device %f", A33220A
   execute commandstr
   execute "GPIBwrite/F=\"VOLTAGE:OFFSET?\\n\""
   execute "GPIBread/T=\"\\n\" junkstring"  //The junkstring takes care of extra terminator characters
   return str2num(junkstring) //in volts
end

Function ClearA33220A() //clears all error messages from 33220A
   NVAR A33220A
   string commandstr
   sprintf commandstr, "GPIB device %f", A33220A
   execute commandstr
   execute "GPIBwrite/F=\"*CLS\""
end

function PrintA33220AError()
   NVAR A33220A
   string/G junkstring
   string commandstr

   sprintf commandstr, "GPIB device %f", A33220A
   execute commandstr
   execute "GPIBwrite/F=\"SYST:ERR?\\n\""
   execute "GPIBread/T=\"\\n\" junkstring"
   print junkstring
end
```

C.4 Yokogawa Voltage Source

This is a set of functions for interfacing with the Yokogawa 7651 DC voltage sources used to control the gate voltages. There are four Yokogawas, indexed 0–3, and the voltage from each is passed through a voltage divider (approximately 2:1) before being applied to the gates. `InitYOKO` is used to initialize the Yokogawas and

other functions turn the output on or off, set the voltage, and set the output range.

```
#pragma rtGlobals=1     // Use modern global access method.

function InitYOKO(yok) //use every time after turning YOKO on
   variable yok //yoko number 0-3
   string/G junkstring = ""
   variable status, n = 0
   string commandstr
   if ((0<=yok) && (yok<=3)) //check for valid yoko number
      NVAR yokonum = $("yoko"+num2istr(yok)) //GPIB address
      sprintf commandstr,"GPIB device %d", yokonum
   else
      print "Invalid Yoko number. Use 0-3. (InitYOKO)"
      return -1
   endif
   execute commandstr
   do
      execute "GPIBwrite/F=\"OC\\n\"" //request status output from YOKO
      execute "GPIBread junkstring" //read output
   while(!StringMatch(junkstring[0,4], "STS1="))
   status = str2num(junkstring[5, Inf])
   make/O/N=8 initYOKObinary = 0
   //parse binary output to check YOKO output on/off flag
   for(n=8;n>0; n-=1)
      if(status>=2^(n-1))
         initYOKObinary[n-1] = 1
         status -= 2^(n-1)
      endif
   endfor
   if(initYOKObinary[4]==1) //bit [4] corresponds to output on/off
      printf "YOKO %d output is ON.  Not changing parameters.  ", yok
      printf "Please turn output off to change.\r"
   else
      printf "YOKO %d output is OFF.\r", yok
      YOKOrange(yok, 5) //set YOKO output range
      printf "YOKO %d range set to 10V scale.\r", yok
   endif
end

function YOKORange(yok,range)
   variable yok //YOKO number
   variable range //2=10mV, 3=100mV, 4=1V, 5=10V, 6=30V

   WAVE/T ExperimentalDetails
   Duplicate/O/R=[][0] ExperimentalDetails, ExperimentalDetailsName

   string commandstr
   if ((0<=yok) && (yok<=3))
      NVAR yokonum = $("yoko"+num2istr(yok))
      sprintf commandstr,"GPIB device %d", yokonum
   else
      print "Invalid Yoko number. Use 0-3. (YOKORange)"
      return -1
   endif

   string fullname = ""
   if ((range >=2) && (range <= 5))
      if(range != round(range))
         printf "%g is not valid for YOKO range: use 2=10mV, 3=100mV, 4=1V, 5=10V, 6=30V.  ", range
         printg "Aborting. (YOKORange)"
         Abort "YOKORange: range not an integer."
      endif
```

```
      string recordvalue = "-"

      fullname = "Vg" + num2istr(yok) + "HardMax"
      FindValue/TEXT=(fullname)/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         sprintf recordvalue, "%g", 10^(range-1) //maximum voltage (in mV) output based on YOKO range
         ExperimentalDetails[V_value][1] = recordvalue //set "Vg#HardMax" in ExperimentalDetails
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", fullname
         printf "Unable to record Yoko maximum. (YOKORange)\r"
      endif

      fullname = "Vg" + num2istr(yok) + "HardMin"
      FindValue/TEXT=(fullname)/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         //minimum voltage (in mV) output based on YOKO range
         sprintf recordvalue, "%g", -10^(range-1)
         ExperimentalDetails[V_value][1] = recordvalue //set "Vg#HardMin" in ExperimentalDetails
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", fullname
         printf "Unable to record Yoko minimum. (YOKORange)\r"
      endif
   elseif(range == 6)
      fullname = "Vg" + num2istr(yok) + "HardMax"
      FindValue/TEXT=(fullname)/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         ExperimentalDetails[V_value][1] = "30000"
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", fullname
         printf "Unable to record Yoko maximum. (YOKORange)\r"
      endif

      fullname = "Vg" + num2istr(yok) + "HardMin"
      FindValue/TEXT=(fullname)/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         ExperimentalDetails[V_value][1] = "-30000"
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", fullname
         printf "Unable to record Yoko minimum. (YOKORange)\r"
      endif
   else
      print "Invalid Yoko range.  Use 2-6. (YOKORange)"
      return -1
   endif

   execute commandstr
   sprintf commandstr, "GPIBwrite \"R%d\"",range
   execute commandstr
   execute "GPIBwrite \"E\""
end

function SetYOKO(yok, setmV) //set YOKO ouput in mV
   variable yok //YOKO number
   variable setmV //Yoko output value in mV
   string commandstr

   NVAR yokonum = $("yoko"+num2istr(yok))
   sprintf commandstr,"GPIB device %d", yokonum
   execute commandstr
   setmV /= 1000 //to set value in mV (value is read by Yoko in Volts)
   sprintf commandstr, "GPIBwrite/F=\"S%f\\n\"",setmV //set YOKO output,
   execute commandstr
   execute "GPIBwrite/F=\"E\\n\""
```

```
end

function ReadYOKO(yok) //returns Yoko output in mV
   variable yok //Yoko number
   string/G junkstring = "not valid"
   string commandstr

   NVAR yokonum = $("yoko"+num2istr(yok))
   sprintf commandstr,"GPIB device %d", yokonum
   execute commandstr
   do
      execute "GPIBwrite/F=\"OD\\n\"" //ask Yoko to send output value
      execute "GPIBread/T= \"\n\" junkstring"
   while(!StringMatch(junkstring[0,3], "NDCV"))
   junkstring = junkstring[4,Inf]
   return str2num(junkstring)*1000 //output in mV
end

function YOKOon(yok) //turn Yoko output on
   variable yok
   string commandstr = ""

   if ((0<=yok) && (yok<=3))
      NVAR yokonum = $("yoko"+num2istr(yok))
      sprintf commandstr,"GPIB device %d", yokonum
   else
      print "Invalid Yoko number. Use 0-3."
      return -1
   endif

   variable outputV = readYOKO(yok)
   if(outputV != 0)
      printf "Yoko %d output voltage nonzero.  Not turning on. (YOKOon)\r", yok
      return -1
   endif

   execute commandstr
   execute "GPIBwrite \"O1\"" //output on
   execute "GPIBwrite \"E\""
end

function YOKOoff(yok) //turn Yoko output off
   variable yok
   string commandstr = ""

   if ((0<=yok) && (yok<=3))
      NVAR yokonum = $("yoko"+num2istr(yok))
      sprintf commandstr,"GPIB device %d", yokonum
   else
      print "Invalid Yoko number. Use 0-3."
      return -1
   endif

   variable outputV = readYOKO(yok)
   if(outputV != 0)
      printf "Yoko %d output voltage nonzero.  Not turning off. (YOKOoff)\r", yok
      return -1
   endif

   execute commandstr
   execute "GPIBwrite \"O0\"" //output off
   execute "GPIBwrite \"E\""
end
```

C.5 Magnet Power Supply

These functions control the magnet power supply. The magnet and power supply are manufactured by Cryogenic Limited and the power supply is SMS series 4.3+. `InitSMS` initializes the power supply and `SMSHeaterOn` and `SMSHeaterOff` turn the persistent switch heater on and off. While ramping the field the persistent switcher heater needs to be on. If the field is to be left stable, then turning the heater off and ramping the power supply current to zero helps reduce helium boil-off. Of course, make sure to ramp the current back to the correct value before turning the heater back on. The function `SMSRampRate` allows the user to set the rate at which the field sweeps and also contains a list of the allowed (discrete) rates.

```
#pragma rtGlobals=1     // Use modern global access method.

function InitSMS() //designed to be run immediately after SMS power on only
   NVAR SMS //GPIB address
   variable/G SMSTPA=0.14031876 //Tesla per Amp conversion factor
   string commandstr
   string/G junkstring = "not valid"
   variable n=0, TtAC = 0

   sprintf commandstr, "GPIB device %d", SMS
   execute commandstr
   execute/Z "GPIBwrite \"gotoremote\""
   execute "GPIBwrite \"gotoremote\"" //for some reason need to do twice to get SMS into remote mode
   //clears startup output lines
   for(n=0; n<23; n+=1)
      execute/Z "GPIBread/T=\"T\" junkstring"
   endfor
   execute "GPIBwrite \"s t\"" //query SMS for Tesla per Amp factor
   execute "GPIBread/T=\"T\" junkstring, junkstring" //read TpA factor
   junkstring = StringFromList(1, junkstring, ":")
   TtAC = str2num(junkstring)
   //check that TpA factor is correct
   if(TtAC==0.14031876)
      print "SMS Tesla to Amp conversion set correctly: 0.14031876 T/A"
   else
      printf "SMS Tesla to Amp converstion incorrect: set at%s\r", junkstring
      print "Correct value is 0.14031876 T/A"
   endif
   //to change the Tesla per Amp conversion to equal the TperA variable use the following two lines:
   //sprintf commandstr, "GPIBwrite \"s t %.8f\"", TperA
   //execute commandstr
   //correct value for Kaster first lab dil fridge is 0.14031876 T/A
   execute "GPIBwrite \"tesla: on\"" //put in Tesla mode
   execute "GPIBwrite \"s mid 0.0\"" //set Mid value to 0 T
   execute "GPIBwrite \"s max 13.8\"" //set Max value to 13.8 T
```

```
   execute "GPIBwrite \"r mid\"" //set SMS to ramp to Mid value

   wait(1)
   WAVE/T ExperimentalDetails
   Duplicate/O/R=[][0] ExperimentalDetails, ExperimentalDetailsName
   FindValue/TEXT=("BRateDefault")/TXOP=4 ExperimentalDetailsName //get default ramp rate
   if(V_Value > -1)
      SetSMSRampRate(str2num(ExperimentalDetails[V_Value][1]))
   else
      Print "\"BRateDefault\" not found in ExperimentalDetails."
      Print "Unable to set magnet ramp rate. (InitSMS)"
   endif
   CheckSMSDirection()

   execute "ni488 ibtmo SMS, 11" //set timeout on SMS to 1 sec. (see NI488 help for info on ibtmo)
end

function SMSCleanBuffer() //cleans SMS output message buffer
   NVAR SMS, V_flag
   string/G junkstring = "not valid"
   string commandstr
   variable n = 0

   sprintf commandstr, "GPIB device %d", SMS
   execute commandstr
   //loop to read messages
   //last one will time out (~15 sec) so don't use this in anything time sensitive
   execute "ni488 ibtmo SMS, 11" //set timeout on SMS to 1 sec. (see NI488 help for info on ibtmo)
   do
      execute "GPIBread/Q/T=\"T̨\" junkstring"
   while(V_flag>0)
end

function SMSRampMid()
   NVAR SMS
   string commandstr

   sprintf commandstr, "GPIB device %d", SMS
   execute commandstr
   execute "GPIBwrite \"r mid\"" //set SMS to ramp to Mid value
end

function SMSPauseOn() //activates Pause feature of SMS
   NVAR SMS
   string commandstr

   sprintf commandstr, "GPIB device %d", SMS
   execute commandstr
   execute "GPIBwrite \"p on\""
end

function SMSPauseOff() //deactivates Pause feature of SMS
   NVAR SMS
   string commandstr

   sprintf commandstr, "GPIB device %d", SMS
   execute commandstr
   execute "GPIBwrite \"p off\""
end

//switches magnet field direction using the reversing switch control
function SetSMSDirection(direction)
   //may only be used when field and current are ZERO!
```

```
      //(magnet will actually sweep to zero and then back to original value
      //(at negative field) when reversing switch command called)
   variable direction //use 1 for positive field, -1 for negative field
   NVAR SMS
   string commandstr

   sprintf commandstr, "GPIB device %d", SMS
   execute commandstr
   if(direction == 1)
      print "Setting magnet direction POSITIVE. (SetSMSDirection)"
      execute "GPIBwrite \"d +\""
   elseif(direction == -1)
      print "Setting magnet direction NEGATIVE. (SetSMSDirection)"
      execute "GPIBwrite \"d -\""
   else
      printf "Argument must be 1 or -1 only; \g is not allowed.  ", direction
      printf "Not changing direction. (SetSMSDirection)\r"
   endif
end

//query SMS and set global SMSDirectionFlag based on reversing switch polarity
function CheckSMSDirection()
   //1: positive, -1: negative
   NVAR SMS, SMSDirectionFlag
   string/G junkstring = "not valid"

   string commandstr
   sprintf commandstr, "GPIB device %d", SMS
   execute commandstr

   variable n = 0
   for(n=0; n<15; n+=1)
      execute "GPIBwrite \"direction\"" //query SMS for heater status
      execute "GPIBread/T=\"T\" junkstring, junkstring" //read heater status message
      if(strsearch(junkstring, "CURRENT DIRECTION:", 0) != -1)
         if(strsearch(junkstring, "POSITIVE", 0) != -1)
            SMSDirectionFlag = 1
            return 1
         elseif(strsearch(junkstring, "NEGATIVE", 0) != -1)
            SMSDirectionFlag = -1
            return -1
         endif
      endif
   endfor
   printf "Unable to determine magnetic field direction after %d attempts.  ", n
   printf "Please retry. (CheckSMSDirection)\r"
   return NaN
end

function SMSHeaterOn() //turn SMS heater on and record time
   NVAR SMS, SMSHeaterOnTime
   string commandstr

   sprintf commandstr, "GPIB device %d", SMS
   execute commandstr
   execute "GPIBwrite \"h on\""
   SMSHeaterOnTime = stopMSTimer(-2)
end

function SMSHeaterOff() //turn SMS heater off
   NVAR SMS
   string commandstr
```

```
   sprintf commandstr, "GPIB device %d", SMS
   execute commandstr
   execute "GPIBwrite \"h off\""
end

//check if SMS heater is on
//returns 0 if off, number of seconds on if on, or -1 if unable to query SMS
function CheckSMSHeater()
   NVAR SMS, SMSHeaterOnTime
   string/G junkstring = "not valid"

   string commandstr
   sprintf commandstr, "GPIB device %d", SMS
   execute commandstr

   variable n = 0
   for(n=0; n<15; n+=1)
      execute "GPIBwrite \"h\"" //query SMS for heater status
      execute "GPIBread/T=\"T\" junkstring, junkstring" //read heater status message
      if(strsearch(junkstring, "HEATER STATUS:", 0) != -1)
         junkstring = StringFromList(1, junkstring, ":")
         junkstring = junkstring[1,2]
         if(StringMatch(junkstring, "ON"))
            //heater is on, return time in seconds it has been on
            return (stopMSTimer(-2)-SMSHeaterOnTime)/1e6
         elseif(StringMatch(junkstring, "OF"))
            return 0 //heater is off
         endif
      endif
   endfor
   printf "Unable to determine heater status after %d attempts.  Please retry. (CheckSMSHeater)\r", n
   return -1
end

function GetSMSField() //query SMS and return magnet field (in mT ), converted from current output
   NVAR SMS, SMSDirectionFlag
   string/G junkstring = "not valid"
   variable oldfield = NaN, newfield = NaN

   string commandstr
   sprintf commandstr, "GPIB device %d", SMS
   execute commandstr

   variable n = 0, m = 0
   for(m=0; m< 5; m+=1)
      for(n=0; n<15; n+=1)
         execute "GPIBwrite \"g o\"" //query SMS for current field
         execute "GPIBread/T=\"T\" junkstring, junkstring" //read field status message
         if(StringMatch(junkstring, "* OUTPUT: * TESLA AT * VOLTS*") == 1)
            junkstring = StringFromList(3, junkstring, ":")
            if(StringMatch(junkstring[1], "-"))
               SMSDirectionFlag = -1
            else
               SMSDirectionFlag = 1
            endif
            newfield = str2num(junkstring)*1000
            break
         endif
      endfor
      if(abs(oldfield - newfield) < 3)
         return newfield
      else
         oldfield = newfield
```

```
      endif
   endfor
   printf "Unable to determine magnetic field after %d attempts.  Please retry. (GetSMSField)\r", m
   return NaN
end

//set SMS Mid value in mT; The field will start to sweep! (assuming Ramp: Mid is set)
function SetSMSMid(field)
   variable field //set value in mT
   field /= 1000 //SMS expects values in T (when in Tesla mode)
   variable/G SMS
   string commandstr

   sprintf commandstr, "GPIB device %d", SMS
   execute commandstr
   sprintf commandstr, "GPIBwrite \"s mid %f\"", field
   execute commandstr
end

function SetSMSRampRate(mTperS) //set ramp rate of SMS
   variable mTperS // Always in mT per second, independent of the tesla on/off setting
   variable maxRampRate = 10 //emergency default in case not defined in ExperimentalDetails (mT/sec.)
   WAVE/T ExperimentalDetails, ExperimentalDetailsName
   variable/G SMSTPA
   variable/G SMS
   string commandstr
   //only a discrete set of 65 ramprates exists in this power supply
   // 0.00120 A/s ~ 0.1684 mT/s
   // 0.00139 A/s ~ 0.1950 mT/s
   // 0.00185 A/s ~ 0.2595 mT/s
   // 0.00213 A/s ~ 0.2989 mT/s
   // 0.00246 A/s ~ 0.3452 mT/s
   // 0.00285 A/s ~ 0.400 mT/s
   // 0.00380 A/s ~ 0.533 mT/s
   // 0.00438 A/s ~ 0.6146 mT/s
   // 0.00506 A/s ~ 0.7100 mT/s
   // 0.00585 A/s ~ 0.8209 mT/s
   // 0.00675 A/s ~ 0.947 mT/s
   // 0.0104  A/s ~ 1.46 mT/s
   // 0.0139  A/s ~ 1.950 mT/s
   // 0.0213  A/s ~ 2.99 mT/s
   // 0.0285  A/s ~ 3.998 mT/s
   // 0.0380  A/s ~ 5.33 mT/s
   // 0.0438  A/s ~ 6.15 mT/s
   // 0.0506  A/s ~ 7.099 mT/s
   // 0.0585  A/s ~ 8.21 mT/s
   // 0.0675  A/s ~ 9.47 mT/s
   // 0.139    A/s ~ 19.5 mT/s
   // 0.380    A/s ~ 53.3 mT/s
   // 0.675    A/s ~ 94.7 mT/s

   //check against maximum ramp rate
   FindValue/TEXT=("BRateMax")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      maxRampRate = str2num(ExperimentalDetails[V_value][1])
   else
      printf "\"BRateMax\" not found in ExperimentalDetails.  ", maxRampRate
      printf "Limiting rate to %g mT/sec. (SetSMSRampRate)\r"
   endif

   if (mTperS<=maxRampRate)
      sprintf commandstr, "GPIB device %d", SMS
      execute commandstr
```

```
      sprintf commandstr, "GPIBwrite \"s r %f\"", mTperS/1000/SMSTPA //set ramp rate
      execute commandstr
   else
      printf "SMS ramp rate too fast: rate must be less than %d mT/s.  ", maxRampRate
      printf "Not changing ramp rate. (SetSMSRampRate)\r"
   endif
end
```

C.6 Digital Multimeter

All dependent variables are read though Hewlett-Packard 34401A digital multimeters (DMM) voltage outputs (using GPIB) and the resulting values then converted into the correct units. Each DMM is initialized with `InitDMM`. Other procedures are used to read the DMM voltage output and to print/clear error messages.

```
#pragma rtGlobals=1     // Use modern global access method.

function InitDMM(dmm) //run every time after turning DMM on
   variable dmm //GPIB address
   string commandstr

   sprintf commandstr, "GPIB device %d", dmm
   execute commandstr

   execute "GPIBwrite/F=\"func \\\"volt:dc\\\"\"" //read DC voltage
   execute "GPIBwrite/F=\"volt:dc:nplc 1\"" //sets integration time
   //number of linecycles: integration time
   //100: 1.67 sec
   //10: 167 msec
   //1: 16.7 msec
   //0.2: 3 msec
   //0.02: 400 microsec.
   execute "GPIBwrite/F=\"volt:dc:rang:auto on\"" //turn autoranging on
   //calibrate zero point once immediately but not continuously
   execute "GPIBwrite/F=\"zero:auto once\""
   DMMLocal(dmm)
End

// This is mostly for diagnostic purposes.  Puts the newest DMM error in global junkstring
function/T GetDMMError(dmm)
   Variable dmm

   string commandstr
   string/G junkstring  // General technique for returning string values from
                  // execute statements without the proliferation of
                  // global variables

   sprintf commandstr, "GPIB device %d", dmm
   execute commandstr

   execute "GPIBwrite/F=\"syst:err?\""
   execute "GPIBread/T=\"\n\" junkstring"
```

```
   DMMLocal(dmm)
   return junkstring
end

// many errorDMM calls flushes any backlog of errors.
function DMMClear(dmm)
   variable dmm
   variable n=0
   string commandstr
   sprintf commandstr, "GPIB device %d", dmm
   execute commandstr

   execute "GPIB deviceclear"
   for(n=0; n<20; n+=1)
      print getdmmerror(dmm)
   endfor
   DMMLocal(dmm)
end

function ReadDMM(dmm) //return DMM value in Volts
   variable dmm
   Variable/G junkvariable
   string commandstr

   sprintf commandstr, "GPIB device %d", dmm
   execute commandstr
   execute "GPIBwrite/F=\"read?\""
   execute "GPIBread/T=\"\n\" junkvariable"

   return junkvariable //returns value in V
End

function DMMLocal(dmm)  //use for sending dmms back to local
   variable dmm
   string commandstr
   sprintf commandstr, "GPIB device %d", dmm
   execute commandstr
   execute "GPIB gotolocal"
end


function AllDMMLocal() //sends DMMs chosen in DataTypeDMM (with Data Types panel) to local
   wave/T DataTypeDMM
   variable maxnumdmms = DimSize(DataTypeDMM, 0)
   variable n = 0
   string dmmnum
   for(n=0; n<maxnumdmms; n+=1)
      if(!StringMatch(DataTypeDMM[n][0], "-")) //check if measurement active
         dmmnum = DataTypeDMM[n][2]
         if(!StringMatch(dmmnum, "0")) //check for valid (nonzero) DMM number
            dmmnum = "dmm" + dmmnum
            NVAR dmm = $dmmnum
            DMMLocal(dmm) //send DMM to local
         endif
      endif
   endfor
end
```

C.7 Data Collection

This section contains functions used to control experimental parameters and to collect data. `SetVal` is used to sweep an independent variable to the desired value. `Do1D` performs a similar sweep and collects data in one-dimensional waves as described in Section C.1. The functions `GateTest` and `Vg1D` are similar but are used to perform specialized tasks. `Do2D` is used to vary two independent variables (a fast axis which is swept continuously and a slow axis which is changed between each fast axis sweep) and collect data in two-dimensional waves. For each of these measurements, the dependent variables to be measured are controlled using the `Data Types` panel, which is created as part of initializing the experiment file. As described in Section C.1, one instance of IGOR is used to collect and save data while another instance is used for simultaneous data analysis. The saved waves are loaded into this second instance using the `Loadibw` function.

```
#pragma rtGlobals=1     // Use modern global access method.

function ScaleFactor(panelnum) //calculates scaling of data based on data type
   variable panelnum
   WAVE/T datatypedmm
   WAVE/T ExperimentalDetails, ExperimentalDetailsName
   variable scale = 1

   FindValue/TEXT=("SampleIac")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      variable sourcecurrent = str2num(ExperimentalDetails[V_value][1])
      if(StringMatch(num2str(sourcecurrent), "NaN"))
         Print "Warning: recorded \"SampleIac\" in ExperimentalDetails not valid. (ScaleFactor)"
      endif
   else
      Print "Warning: \"SampleIac\" not found in ExperimentalDetails. (ScaleFactor)"
   endif
   FindValue/TEXT=("SampleVac")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      variable sourcevoltage = str2num(ExperimentalDetails[V_value][1])
      if(StringMatch(num2str(sourcecurrent), "NaN"))
         Print "Warning: recorded \"SampleVac\" in ExperimentalDetails not valid. (ScaleFactor)"
      endif
   else
      Print "Warning: \"SampleVac\" not found in ExperimentalDetails. (ScaleFactor)"
   endif
   FindValue/TEXT=("IdcStop")/TXOP=4 ExperimentalDetailsName
```

```
   if(V_Value > -1)
      variable IDC = str2num(ExperimentalDetails[V_value][1])
      if(StringMatch(num2str(IDC), "NaN"))
         Print "Warning: recorded \"IdcStop\" in ExperimentalDetails not valid. (ScaleFactor)"
      endif
   else
      Print "Warning: \"IdcStop\" not found in ExperimentalDetails. (ScaleFactor)"
   endif
   FindValue/TEXT=("AV47Range")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      variable av47range = str2num(ExperimentalDetails[V_value][1])
      if(StringMatch(num2str(av47range), "NaN"))
         Print "Warning: recorded \"AV47Range\" in ExperimentalDetails not valid. (ScaleFactor)"
      endif
   else
      Print "Warning: \"AV47Range\" not found in ExperimentalDetails. (ScaleFactor)"
   endif

   strswitch(datatypedmm[panelnum][0])
      case "Rxx":
      case "Rxy":
      case "Rl":
      case "Rd":
      case "Rdp":
      case "Rdm":
         scale *= 1/(sourcecurrent*1e-9*CR0()) //R = V/(A * h/e^2)
         break
      case "RxxDC":
      case "RxyDC":
      case "RdDC":
         scale *= 1/(IDC*1e-9*CR0())
         break
      case "G":
      case "Gxx":
      case "Gxy":
      case "Gl":
      case "Gd":
         scale *= -CR0()/(sourcevoltage*1e-6) //G = A/(V * e^2/h)
         break
      case "Bmeas":
         scale *= 100/1.3923*3000/3017.5
         break
      case "Tmc":
         scale *= av47range/2
         break
   endswitch

   return scale
end

//sweeps an independent variable to value, calls on proper sweep function for each ivar
function SetVal(ivar, value, [rate])
   string ivar //variable to sweep
   variable value, rate //sweep to value at rate
   wave/T InVarWave, ExperimentalDetails, ExperimentalDetailsName
   string searchstr

   string recordstr = ""
   sprintf recordstr, "%s: %s: SetVal(%s, %g, [%g])\r", date(), time(), ivar, value, rate
   Notebook Record, selection={StartofFile, StartofFile}, text = recordstr
   SaveNotebook/O/P=savepath/S=6 Record as "Record.txt"

   //if Vg specified, create basic Vg identifier and calculate number of gates
```

```
if(StringMatch(ivar[0,1], "Vg"))
   searchstr = "Vg"
   variable numgates = strlen(ivar)
   variable n = 2
   string yokostr = ""
   variable stepGate = 0.5
   FindValue/TEXT=("VgStep")/TXOP=4 ExperimentalDetailsName
   if(V_value > -1)
      stepGate = str2num(ExperimentalDetails[V_value][1]) //voltage step size at device
   else
      Print "Warning: \"VgStep\" not found in ExperimentalDetails."
      Print "Using step size of 0.5 mV. (SetVal)"
   endif
elseif(StringMatch(ivar, "Idc")) //get VtoI R for Vdc to Idc conversion
   searchstr = ivar
   FindValue/TEXT=("VtoI R")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      variable VtoIR = str2num(ExperimentalDetails[V_value][1])
   else
      Print "\"VtoI R\" not found in ExperimentalDetails.  Aborting. (SetVal)"
      Abort "SetVal: cannot find VtoI R in ExperimentalDetails."
   endif
elseif(StringMatch(ivar, "Vdc")) //get DC voltage divider
   searchstr = ivar
   FindValue/TEXT=("VdcDivider")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      variable VDivDC = str2num(ExperimentalDetails[V_value][1])
   else
      Print "\"VdcDivider\" not found in ExperimentalDetails.  Aborting. (SetVal)"
      Abort "SetVal: cannot find VdcDivider in ExperimentalDetails."
   endif
else
   searchstr = ivar
endif

FindValue/TEXT=searchstr/TXOP=4 InVarWave //check that ivar is a valid independent variable
if((V_value == -1) || (V_value >= DimSize(InVarWave, 0)))
   printf "Independent variable %s not in InVarWave.  Aborting. (SetVal)\r", searchstr
   Abort "SetVal: independent variable not valid."
else
   string ivarunits = InVarWave[V_value][1]
endif

if(ParamIsDefault(rate) || rate == 0)
   if(StringMatch(searchstr, "Vg")) //use default rate of first gate number specified
      yokostr = "Vg" + ivar[2]
      FindValue/TEXT=(yokostr + "RateDefault")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         rate = str2num(ExperimentalDetails[V_value][1])
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokostr + "RateDefault"
         printf "Aborting. (SetVal)\r"
         Abort "SetVal: no default sweep rate defined."
      endif
   else //set rate to default rate for generic variable
      FindValue/TEXT=(searchstr + "RateDefault")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         rate = str2num(ExperimentalDetails[V_value][1])
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "RateDefault"
         printf "Aborting. (SetVal)\r"
         Abort "SetVal: no default sweep rate defined."
      endif
```

```
   endif
endif
rate = abs(rate) //make sure rate is positive

if(StringMatch(searchstr, "B")) //check if heater has been on long enough
   variable timeon = CheckSMSHeater()
   if(timeon==0)
      Print "SMS heater not on!  Unable to sweep B.  Aborting. (SetVal)"
      Abort "SetVal: SMS heater not on."
   elseif(timeon==-1)
      Print "SMS heater may not be on.  Make sure heater is on and retry.  Aborting. (SetVal)"
      Abort "SetVal: error from CheckSMSHeater."
   elseif(timeon<300) //heater must be on for at least 5 minutes
      printf "SMS heater turned on too recently.  Wait %d seconds and try again.  ", 300-timeon
      printf "Aborting. (SetVal)\r"
      Abort "SetVal: SMS heater not on long enough."
   endif
endif

variable maxval = 0, minval = 0, maxrate = 0
string recordvalue
//check inputs against Max, Min, and RateMax values; update current Stop and Rate values
if(StringMatch(searchstr, "Vg"))
   variable divider = Inf
   //actual sweep rate to account for sweeping multiple gates
   variable actualrate = rate * (numgates - 2)
   for(n=2; n<numgates; n+=1) //step through each gate number
      yokostr = "Vg" + ivar[n]

      FindValue/TEXT=(yokostr + "Divider")/TXOP=4 ExperimentalDetailsName //get divider value
      if(V_Value > -1)
         if(str2num(ExperimentalDetails[V_value][1]) > 0)
            divider = str2num(ExperimentalDetails[V_value][1])
         else
            printf "\"%s\" value not positive.  Aborting. (SetVal)\r", yokostr + "Divider"
            Abort "SetVal: Vg#Divider not positive."
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokostr + "Divider"
         printf "Aborting. (SetVal)\r"
         Abort "SetVal: cannot find Vg#Divider in ExperimentalDetails."
      endif
      //check hardware (Yoko) maximum
      FindValue/TEXT=(yokostr + "HardMax")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         variable hardmax = str2num(ExperimentalDetails[V_value][1])
         if((value*divider)>hardmax)
            printf "Specified value %g V too high.  ", value*divider, hardmax
            printf "Yoko limited to %g V.  Aborting. (SetVal)\r"
            Abort "SetVal: value exceeds hardware maximum."
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokostr + "HardMax"
         printf "Unable to check if value exceeds maximum. (SetVal)\r"
      endif
      //check experimental (device) maximum
      FindValue/TEXT=(yokostr + "ExpMax")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         variable expmax = str2num(ExperimentalDetails[V_value][1])
         if(value>expmax)
            printf "Specified value %g %s too high.  ", value, ivarunits, expmax, ivarunits
            printf "Limited to %g %s.  Aborting. (SetVal)\r"
            Abort "SetVal: value exceeds maximum."
```

```
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokostr + "ExpMax"
         printf "Unable to check if value exceeds maximum. (SetVal)\r"
      endif
         //check hardware (Yoko) minimum
         FindValue/TEXT=(yokostr + "HardMin")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         variable hardmin = str2num(ExperimentalDetails[V_value][1])
         if((value*divider)<hardmin)
            printf "Specified value %g V too low.  ", value*divider, hardmin
            printf "Yoko limited to %g V.  Aborting. (SetVal)\r"
            Abort "SetVal: value exceeds hardware minimum"
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokostr + "HardMin"
         printf "Unable to check if value exceeds minimum. (SetVal)\r"
      endif
      //check experimental (device) minimum
      FindValue/TEXT=(yokostr + "ExpMin")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         variable expmin = str2num(ExperimentalDetails[V_value][1])
         if(value<expmin)
            printf "Specified value %g %s too low.  ", value, ivarunits, expmin, ivarunits
            printf "Limited to %g %s.  Aborting. (SetVal)\r"
            Abort "SetVal: value exceeds minimum"
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokostr + "ExpMin"
         printf "Unable to check if value exceeds minimum. (SetVal)\r"
      endif

      FindValue/TEXT=(yokostr + "RateMax")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         maxrate = str2num(ExperimentalDetails[V_value][1])
         if(actualrate>maxrate)
            actualrate = maxrate
            printf "Actual rate too fast: "
            printf "reducing actual rate to %g %s. (SetVal)\r", maxrate, ivarunits + "/sec"
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokostr + "RateMax"
         printf "Sweep rate may be too fast. (SetVal)\r"
      endif

      FindValue/TEXT=(yokostr + "Stop")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         sprintf recordvalue, "%g", value
         ExperimentalDetails[V_value][1] = recordvalue
      else
         printf "\"%s\" not found in ExperimentalDetails. ", yokostr + "Stop"
         printf "Unable to record stop value. (SetVal)\r"
      endif
      FindValue/TEXT=(yokostr + "Rate")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         sprintf recordvalue, "%g", actualrate
         ExperimentalDetails[V_value][1] = recordvalue
      else
         printf "\"%s\" not found in ExperimentalDetails. ", yokostr + "Rate"
         printf "Unable to record sweep rate. (SetVal)\r"
      endif
   endfor
elseif(StringMatch(searchstr, "Idc"))
```

```
    FindValue/TEXT=(searchstr + "Max")/TXOP=4 ExperimentalDetailsName
    if(V_Value > -1)
       maxval = str2num(ExperimentalDetails[V_value][1])
       if((value*1e-9*VtoIR)>maxval)
          printf "Specified value %g nA too high.  ", value
          printf "33220A limited to %g V.  Aborting. (SetVal)\r", maxval
          Abort "SetVal: value exceeds hardware maximum."
       endif
    else
       printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Max"
       printf "Unable to check if value exceeds maximum. (SetVal)\r"
    endif
    FindValue/TEXT=(searchstr + "Min")/TXOP=4 ExperimentalDetailsName
    if(V_Value > -1)
       minval = str2num(ExperimentalDetails[V_value][1])
       if((value*1e-9*VtoIR)<minval)
          printf "Specified value %g nA too low.  ", value
          printf "33220A limited to %g V.  Aborting. (SetVal)\r", minval
          Abort "SetVal: value exceeds hardware minimum."
       endif
    else
       printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Min"
       printf "Unable to check if value exceeds minimum. (SetVal)\r"
    endif
    FindValue/TEXT=(searchstr + "RateMax")/TXOP=4 ExperimentalDetailsName
    if(V_Value > -1)
       maxrate = str2num(ExperimentalDetails[V_value][1])
       if(rate>maxrate)
          rate = maxrate
          printf "Rate too fast: reducing rate to %g %s. (SetVal)\r", maxrate, ivarunits + "/sec"
       endif
    else
       printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "RateMax"
       prtinf "Sweep rate may be too fast. (SetVal)\r"
    endif

    FindValue/TEXT=(searchstr + "Stop")/TXOP=4 ExperimentalDetailsName
    if(V_Value > -1)
       sprintf recordvalue, "%g", value
       ExperimentalDetails[V_value][1] = recordvalue
    else
       printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Stop"
       printf "Unable to record stop value. (SetVal)\r"
    endif
    FindValue/TEXT=(searchstr + "Rate")/TXOP=4 ExperimentalDetailsName
    if(V_Value > -1)
       sprintf recordvalue, "%g", rate
       ExperimentalDetails[V_value][1] = recordvalue
    else
       printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Rate"
       printf "Unable to record sweep rate. (SetVal)\r"
    endif
 elseif(StringMatch(searchstr, "Vdc"))
    FindValue/TEXT=(searchstr + "Max")/TXOP=4 ExperimentalDetailsName
    if(V_Value > -1)
       maxval = str2num(ExperimentalDetails[V_value][1])
       if((value*1e-3*VDivDC)>maxval)
          printf "Specified value %g mV too high.  ", value
          printf "33220A limited to %g V.  Aborting. (SetVal)\r", maxval
          Abort "SetVal: value exceeds hardware maximum."
       endif
    else
       printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Max"
```

```
      printf "Unable to check if value exceeds maximum. (SetVal)\r"
   endif
   FindValue/TEXT=(searchstr + "Min")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      minval = str2num(ExperimentalDetails[V_value][1])
      if((value*1e-3*VDivDC)<minval)
         printf "Specified value %g mV too low.  ", value
         printf "33220A limited to %g V.  Aborting. (SetVal)\r", minval
         Abort "SetVal: value exceeds hardware minimum."
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Min"
      printf "Unable to check if value exceeds minimum. (SetVal)\r"
   endif
   FindValue/TEXT=(searchstr + "RateMax")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      maxrate = str2num(ExperimentalDetails[V_value][1])
      if(rate>maxrate)
         rate = maxrate
         printf "Rate too fast: reducing rate to %g %s. (SetVal)\r", maxrate, ivarunits + "/sec"
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "RateMax"
      printf "Sweep rate may be too fast. (SetVal)\r"
   endif

   FindValue/TEXT=(searchstr + "Stop")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      sprintf recordvalue, "%g", value
      ExperimentalDetails[V_value][1] = recordvalue
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Stop"
      printf "Unable to record stop value. (SetVal)\r"
   endif
   FindValue/TEXT=(searchstr + "Rate")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      sprintf recordvalue, "%g", rate
      ExperimentalDetails[V_value][1] = recordvalue
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Rate"
      printf "Unable to record sweep rate. (SetVal)\r"
   endif
else
   FindValue/TEXT=(searchstr + "Max")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      maxval = str2num(ExperimentalDetails[V_value][1])
      if(value>maxval)
         printf "Specified value %g %s too high.  ", value, ivarunits
         printf "Limited to %g %s.  Aborting. (SetVal)\r", maxval, ivarunits
         Abort "SetVal: value exceeds maximum."
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Max"
      printf "Unable to check if value exceeds maximum. (SetVal)\r"
   endif
   FindValue/TEXT=(searchstr + "Min")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      minval = str2num(ExperimentalDetails[V_value][1])
      if(value<minval)
         printf "Specified value %sg %s too low.  ", value, ivarunits
         printf "Limited to %g %s.  Aborting. (SetVal)\r", minval, ivarunits
         Abort "SetVal: value exceeds minimum."
      endif
```

```
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Min"
         printf "Unable to check if value exceeds minimum. (SetVal)\r"
      endif
      FindValue/TEXT=(searchstr + "RateMax")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         maxrate = str2num(ExperimentalDetails[V_value][1])
         if(rate>maxrate)
            rate = maxrate
            printf "Rate too fast: reducing rate to %g %s. (SetVal)\r", maxrate, ivarunits + "/sec"
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "RateMax"
         printf "Sweep rate may be too fast. (SetVal)\r"
      endif

      FindValue/TEXT=(searchstr + "Stop")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         sprintf recordvalue, "%g", value
         ExperimentalDetails[V_value][1] = recordvalue
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Stop"
         printf "Unable to record stop value. (SetVal)\r"
      endif
      FindValue/TEXT=(searchstr + "Rate")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         sprintf recordvalue, "%g", rate
         ExperimentalDetails[V_value][1] = recordvalue
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Rate"
         printf "Unable to record sweep rate. (SetVal)\r"
      endif
   endif
   DoUpdate

   variable currentval = NaN, starttime = stopMSTimer(-2)
   variable k = 1
   try
      if(StringMatch(searchstr, "B")) //sweep B
//       CheckSMSDirection()  //uncomment if allowing sweeps through zero field
//       NVAR SMSDirectionFlag
         SetSMSRampRate(rate) //rate in mT/sec
//       if((sign(SMSDirectionFlag) != sign(value)) && (value != 0))
//          print "Warning: Magnetic field direction will be switched!"
//          print "Please wait or manually abort. (SetVal)"
//          variable startB = GetSMSField()
//          SetSMSMid(0)
//          currentval = 0
//          wait(abs(startB)/rate)
//          wait(2)
//          SetSMSDirection(sign(value))
//          wait(3)
//       endif
         wait(.1)
         SetSMSMid(abs(value)) //value in mT
         currentval = value
      elseif(StringMatch(searchstr, "Vg")) //sweep Vg (each gate one at a time)
         variable yokonum = -1
         string fullname
         for(n=2; n<numgates; n+=1)
            yokonum = str2num(ivar[n])
            fullname = "Vg" + num2istr(yokonum) + "Divider"
            FindValue/TEXT=(fullname)/TXOP=4 ExperimentalDetailsName
```

```
      if(V_Value > -1)
         divider = str2num(ExperimentalDetails[V_value][1])
      else
         printf "\%s\" not found in ExperimentalDetails.  Aborting. (SetVal)\r", fullname
         Abort "SetVal: cannot find YokoDivider in ExperimentalDetails."
      endif
      variable startVg = ReadYOKO(yokonum) //current Yoko output
      variable endVg = value*divider //target Yoko output
      variable stepVg = stepGate * divider //Yoko step size
      //The Yokos seem to have about a 0.01 sec delay per point.
      //This combined with the stepVg value gives an effective maximum rate.
      variable numstepsg = round(abs(endVg-startVg)/stepVg) //number of steps
      if(numstepsg <1)
         numstepsg = 1
      endif
      stepVg = (endVg-startVg)/numstepsg //recalculate exact step size based on number of steps
      variable steptimeg = abs(1e6*stepVg/(actualrate*divider)) //time per step in microseconds
      variable waittimeg = 0
      for(k=1; k<numstepsg + 1; k+=1) //step voltage to target
         waittimeg = stopMSTimer(-2)
         SetYOKO(yokonum, startVg + stepVg*k)
         currentval = (startVg + stepVg*k)/divider
         do
         //delay after each step to control sweep rate
         while((stopMSTimer(-2)-waittimeg)<steptimeg)
      endfor
      SetYOKO(yokonum, endVg) //avoid any rounding problems at end
      currentval = value
   endfor
elseif(StringMatch(searchstr, "Time")) //'sweep' in time
   wait(1/rate)
elseif(StringMatch(searchstr, "Idc")) //sweep Idc
   variable startVi = ReadA33220ADC() //current 33220A Vdc
   variable endVi = value*VtoIR*1e-9 //target Vdc
   variable stepVi = 0.01 //step size in Volts
   variable numstepsi = round(abs(endVi-startVi)/stepVi) //number of steps
   if(numstepsi <1)
      numstepsi = 1
   endif
   stepVi = (endVi-startVi)/numstepsi //recalculate exact step size based on number of steps
   variable steptimei = abs(1e6*stepVi/(rate*1e-9*VtoIR)) //time per step in microseconds
   variable waittimei = 0
   for(k=1; k<numstepsi + 1; k+=1) //step voltage to target
      waittimei = stopMSTimer(-2)
      SetA33220ADC(startVi + stepVi*k)
      currentval = (startVi + stepVi*k)*1e9/VtoIR
      do
      while((stopMSTimer(-2)-waittimei)<steptimei) //delay after each step to control sweep rate
   endfor
   SetA33220ADC(endVi)
   currentval = value
elseif(StringMatch(searchstr, "Vdc")) //sweep Vdc
   variable startV = ReadA33220ADC() //current 33220A Vdc
   variable endV = value*VDivDC*1e-3 //target Vdc
   variable stepV = 0.01 //step size in Volts
   variable numsteps = round(abs(endV-startV)/stepV) //number of steps
   if(numsteps <1)
      numsteps = 1
   endif
   stepV = (endV-startV)/numsteps //recalculate exact step size based on number of steps
   variable steptime = abs(1e6*stepV/(rate*1e-3*VDivDC)) //time per step in microseconds
   variable waittime = 0
   for(k=1; k<numsteps + 1; k+=1) //step voltage to target
```

```
            waittime = stopMSTimer(-2)
            SetA33220ADC(startV + stepV*k)
            currentval = (startV + stepV*k)*1e3/VDivDC
            do
            while((stopMSTimer(-2)-waittime)<steptime) //delay after each step to control sweep rate
         endfor
         SetA33220ADC(endV)
         currentval = value
      endif
   catch
   endtry
   try //continue to updating ExperimentalDetails even if SetVal aborted
      //update Start, Stop, and Rate to current values
      //(may be different from inputs if function was aborted)
      if(StringMatch(searchstr, "Vg"))
         for(n=2; n<numgates; n+=1)
            yokostr = "Vg" + ivar[n]
            FindValue/TEXT=(yokostr+"Start")/TXOP=4 ExperimentalDetailsName
            if(V_Value > -1)
               sprintf recordvalue, "%g", currentval
               ExperimentalDetails[V_value][1] = recordvalue
            else
               printf "\"%s\" not found in ExperimentalDetails.  ", yokostr + "Start"
               printf "Unable to record update after sweep. (SetVal)\r"
            endif
            FindValue/TEXT=(yokostr+"Stop")/TXOP=4 ExperimentalDetailsName
            if(V_Value > -1)
               sprintf recordvalue, "%g", currentval
               ExperimentalDetails[V_value][1] = recordvalue
            else
               printf "\"%s\" not found in ExperimentalDetails.  ", yokostr + "Stop"
               printf "Unable to record update after sweep. (SetVal)\r"
            endif
            FindValue/TEXT=(yokostr+"Rate")/TXOP=4 ExperimentalDetailsName
            if(V_Value > -1)
               ExperimentalDetails[V_value][1] = "0"
            else
               printf "\"%s\" not found in ExperimentalDetails.  ", yokostr + "Rate"
               printf "Unable to update after sweep. (SetVal)\r"
            endif
         endfor
      else
         FindValue/TEXT=(searchstr+"Start")/TXOP=4 ExperimentalDetailsName
         if(V_Value > -1)
            sprintf recordvalue, "%g", currentval
            ExperimentalDetails[V_value][1] = recordvalue
         else
            printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Start"
            printf "Unable to record update after sweep. (SetVal)\r"
         endif
         FindValue/TEXT=(searchstr+"Stop")/TXOP=4 ExperimentalDetailsName
         if(V_Value > -1)
            sprintf recordvalue, "%g", currentval
            ExperimentalDetails[V_value][1] = recordvalue
         else
            printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Stop"
            printf "Unable to record update after sweep. (SetVal)\r"
         endif
         FindValue/TEXT=(searchstr+"Rate")/TXOP=4 ExperimentalDetailsName
         if(V_Value > -1)
            ExperimentalDetails[V_value][1] = "0"
         else
            printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Rate"
```

```
            printf "Unable to update after sweep. (SetVal)\r"
         endif
      endif
   catch
      Print "Warning: function aborted, ExperimentalDetails may not be properly updated. (SetVal)"
   endtry
   variable endtime = ((stopMSTimer(-2)-starttime)*1e-6/60)
   printf "Sweep finished at %s, elapsed time %6.3f min.\r",time(),endtime
   sprintf recordstr, "Sweep finished at %s, elapsed time %6.3f min.\r",time(),endtime
   Notebook Record, selection={StartofFile, StartofFile}, text = recordstr
   SaveNotebook/O/P=savepath/S=6 Record as "Record.txt"
end

function MeasureDMM(panelnum) //return scaled data output based on datatypes panel
   variable panelnum  //number listed on datatypes panel; NOT dmm number
   panelnum -= 1

   initgpib()

   WAVE/T datatypedmm
   WAVE/T ExperimentalDetails
   Duplicate/O/R=[][0] ExperimentalDetails, ExperimentalDetailsName
   variable result
   string dmmnum = datatypedmm[panelnum][2] //get dmm number from panel

   if(StringMatch(datatypedmm[panelnum][0], "-"))
      printf "No data type chosen for panel number %d.  Aborting. (MeasureDMM)\r", panelnum
      Abort "MeasureDMM: no data type chosen."
   endif

   if(StringMatch(dmmnum,"0"))
      printf "DMM number not set for panel number %d.  ", panelnum
      printf "Cannot measure DMM.  Aborting. (MeasureDMM)\r"
      Abort "MeasureDMM: DMM number not set."
   endif

   variable scale = 1 //for ReadDMM returning V
   variable offset = 0
   FindValue/TEXT=("DMMoffset" + dmmnum)/TXOP=4 ExperimentalDetailsName //get dmm offset (y-intercept)
   if(V_Value > -1)
      offset += str2num(ExperimentalDetails[V_value][1])
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", "DMMoffset" + dmmnum
      printf "Offset not changed. (MeasureDMM)\r"
   endif
   FindValue/TEXT=("DMMscale" + dmmnum)/TXOP=4 ExperimentalDetailsName //get dmm scale factor
   if(V_Value > -1)
      scale *= str2num(ExperimentalDetails[V_Value][1])
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", "DMMscale" + dmmnum
      printf "Scale not changed. (MeasureDMM)\r"
   endif
   if(!stringmatch(datatypedmm[panelnum][4], "0")) //get lock-in sensitivity
      FindValue/TEXT=("LIsens" + datatypedmm[panelnum][4])/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         scale *= str2num(ExperimentalDetails[V_value][1])/10
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", "LIsens" + datatypedmm[panelnum][3]
         printf "Scale not changed. (MeasureDMM)\r"
      endif
   endif
   if(!stringmatch(datatypedmm[panelnum][5], "0")) // get pre-amp amplification
      FindValue/TEXT=("PAamp" + datatypedmm[panelnum][5])/TXOP=4 ExperimentalDetailsName
```

```
      if(V_Value > -1)
         scale /= str2num(ExperimentalDetails[V_value][1])
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", "PAamp" + datatypedmm[panelnum][4]
         printf "Scale not changed. (MeasureDMM)\r"
      endif
   endif

   scale *= ScaleFactor(panelnum)
   dmmnum = "dmm" + dmmnum
   NVAR dmm = $dmmnum

   //constants for mixing chamber temperature fit
   variable A0 = 411.7925271369486, A1 = -487.3966692536534, A2 = 233.1118681633504
   variable A3 = -55.85454979733308, A4 = 6.690672513604219, A5 = -0.3205752792140923

   result = scale*(readdmm(dmm) - offset) //measure and rescale dmm output
   DMMLocal(dmm) //return dmm to local mode
   if(stringmatch(datatypedmm[panelnum][0], "Tmc")) //implement nonlinear temperature fit
      result = A0 + A1 * log(result) + A2*(log(result))^2 + A3*(log(result))^3
      result += A4*(log(result))^4 + A5*(log(result))^5
      result = 10^result
   endif

   return result
end

function TakeData(datawave) //measures each dmm to get data points in a set of indexed waves
   //wave to store data in; should be of same length
   //as number of simultaneous measurement choices allowed in datatypes panel
   wave datawave

   wave/T datatypedmm
   variable maxmeasurements = DimSize(datatypedmm, 0)
   variable n = 0

   for(n=0; n<maxmeasurements; n +=1)
      if(!stringmatch(datatypedmm[n][0],"-"))
         datawave[n] = readdmm(str2num(datatypedmm[n][3]))
      endif
   endfor
end

//sweeps an independent variable to value, calls on proper sweep function for each ivar
function SweepVal(ivar, value, rate, startB)
   //For use in Do1D or Do2D ONLY!  Does not include important safety checks (like magnet heater on).
   string ivar //variable to sweep
   //sweep to value at rate, startB is current magnetic field (only matters when sweeping B)
   variable value, rate, startB
   wave/T InVarWave, ExperimentalDetails, ExperimentalDetailsName
   string searchstr

   //if Vg specified, create basic Vg identifier and calculate number of gates
   if(StringMatch(ivar[0,1], "Vg"))
      searchstr = "Vg"
      variable numgates = strlen(ivar)
      variable n = 2
      string yokostr = ""
      variable stepGate = 0.5
      FindValue/TEXT=("VgStep")/TXOP=4 ExperimentalDetailsName
      if(V_value > -1)
         stepGate = str2num(ExperimentalDetails[V_value][1]) //voltage step size at device
      else
```

```
         Print "Warning: \"VgStep\" not found in ExperimentalDetails."
         Print "Using step size of 0.5 mV. (SweepVal)"
      endif
   elseif(StringMatch(ivar, "Idc")) //get VtoI R for Vdc to Idc conversion
      searchstr = ivar
      FindValue/TEXT=("VtoI R")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         variable VtoIR = str2num(ExperimentalDetails[V_value][1])
      else
         Print "\"VtoI R\" not found in ExperimentalDetails.  Aborting. (SweepVal)"
         Abort "SweepVal: cannot find VtoI R in ExperimentalDetails."
      endif
   elseif(StringMatch(ivar, "Vdc")) //get DC voltage divider
      searchstr = ivar
      FindValue/TEXT=("VdcDivider")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         variable VDivDC = str2num(ExperimentalDetails[V_value][1])
      else
         Print "\"VdcDivider\" not found in ExperimentalDetails.  Aborting. (SweepVal)"
         Abort "SweepVal: cannot find VtoI R in ExperimentalDetails."
      endif
   else
      searchstr = ivar
   endif
   if(rate == 0)
      if(StringMatch(searchstr, "Vg")) //use default rate of first gate number specified
         yokostr = "Vg" + ivar[2]

         FindValue/TEXT=(yokostr + "RateDefault")/TXOP=4 ExperimentalDetailsName
         if(V_Value > -1)
            rate = abs(str2num(ExperimentalDetails[V_value][1]))
         else
            printf "\"%s\" not found in ExperimentalDetails.  ", yokostr + "RateDefault"
            printf "Aborting. (SweepVal)\r"
            Abort "SweepVal: no default sweep rate defined."
         endif
      else //set rate to default rate for generic variable
         FindValue/TEXT=(searchstr + "RateDefault")/TXOP=4 ExperimentalDetailsName
         if(V_Value > -1)
            rate = abs(str2num(ExperimentalDetails[V_value][1]))
         else
            printf "\"%s\" not found in ExperimentalDetails.  ", yokostr + "RateDefault"
            printf "Aborting. (SweepVal)\r"
            Abort "SweepVal: no default sweep rate defined."
         endif
      endif
   else
      rate = abs(rate) //make sure rate is positive
   endif

   variable k = 1
   try
      if(StringMatch(searchstr, "B")) //sweep B
         SetSMSRampRate(rate) //rate in mT/sec
//       variable zerosweepflag = 0 //uncomment if allowing sweeps through zero field
//       if(startB == 0)
//          CheckSMSDirection()
//          NVAR SMSDirectionFlag
//          if(sign(SMSDirectionFlag) != sign(value))
//             print "Alert: switching magnetic field direction. (SweepVal)"
//             SetSMSMid(0)
//             currentval = 0
//             wait(2)
```

```
//             SetSMSDirection(sign(value))
//             wait(3)
//          endif
//       elseif((sign(startB) != sign(value)) && (value != 0))
//          print "Alert: switching magnetic field direction. (SweepVal)"
//          zerosweepflag = 1
//          SetSMSMid(0)
//          currentval = 0
//          wait(abs(startB)/rate)
//          wait(2)
//          SetSMSDirection(sign(value))
//          wait(3)
//       endif
         SetSMSMid(abs(value)) //value in mT
         SMSRampMid()
//       if(zerosweepflag == 0)
            wait(abs(value - startB)/rate)
//       elseif(zerosweepflag == 1)
//          wait(abs(value)/rate)
//       else
//          printf "Variable zerosweepflag has invalid value %g.  ", zerosweepflag
//          printf "Aborting. (SweepVal)\r"
//          Abort "SweepVal: invalid zerosweepflag."
//       endif
      elseif(StringMatch(searchstr, "Vg")) //sweep Vg (each gate one at a time)
         variable yokonum = -1
         string fullname
         for(n=2; n<numgates; n+=1)
            yokonum = str2num(ivar[n])
            fullname = "Vg" + num2istr(yokonum) + "Divider"
            FindValue/TEXT=(fullname)/TXOP=4 ExperimentalDetailsName
            if(V_Value > -1)
               variable divider = str2num(ExperimentalDetails[V_value][1])
            else
               printf "\%s\" not found in ExperimentalDetails.  Aborting. (SweepVal)\r", fullname
               Abort "SweepVal: cannot find YokoDivider in ExperimentalDetails."
            endif
            variable startVg = ReadYOKO(yokonum) //current Yoko output
            variable endVg = value*divider //target Yoko output
            variable stepVg = stepGate*divider //Yoko step size
            //The Yokos seem to have about a 0.01 sec delay per point.
            //This combined with the stepVg value gives an effective maximum rate.
            variable numstepsg = round(abs(endVg-startVg)/stepVg) //number of steps
            if(numstepsg <1)
               numstepsg = 1
            endif
            stepVg = (endVg-startVg)/numstepsg //recalculate exact step size based on number of steps
            variable steptimeg = abs(1e6*stepVg/(rate*divider)) //time per step in microseconds
            variable waittimeg = 0
            for(k=1; k<numstepsg + 1; k+=1) //step voltage to target
               waittimeg = stopMSTimer(-2)
               SetYOKO(yokonum, startVg + stepVg*k)
               do
               //delay after each step to control sweep rate
               while((stopMSTimer(-2)-waittimeg)<steptimeg)
            endfor
            SetYOKO(yokonum, endVg) //avoid any rounding problems at end
         endfor
      elseif(StringMatch(searchstr, "Time")) //'sweep' in time
         wait(1/rate)
      elseif(StringMatch(searchstr, "Idc")) //sweep Idc
         variable startVi = ReadA33220ADC() //current 33220A Vdc
         variable endVi = value*VtoIR*1e-9 //target Vdc
```

```
         variable stepVi = 0.01 //step size in Volts
         variable numstepsi = round(abs(endVi-startVi)/stepVi) //number of steps
         if(numstepsi <1)
            numstepsi = 1
         endif
         stepVi = (endVi-startVi)/numstepsi //recalculate exact step size based on number of steps
         variable steptimei = abs(1e6*stepVi/(rate*1e-9*VtoIR)) //time per step in microseconds
         variable waittimei = 0
         for(k=1; k<numstepsi + 1; k+=1) //step voltage to target
            waittimei = stopMSTimer(-2)
            SetA33220ADC(startVi + stepVi*k)
            do
            //delay after each step to control sweep rate
            while((stopMSTimer(-2)-waittimei)<steptimei)
         endfor
         SetA33220ADC(endVi)
      elseif(StringMatch(searchstr, "Vdc")) //sweep Vdc
         variable startV = ReadA33220ADC() //current 33220A Vdc
         variable endV = value*VDivDC*1e-3 //target Vdc
         variable stepV = 0.01 //step size in Volts
         variable numsteps = round(abs(endV-startV)/stepV) //number of steps
         if(numsteps <1)
            numsteps = 1
         endif
         stepV = (endV-startV)/numsteps //recalculate exact step size based on number of steps
         variable steptime = abs(1e6*stepV/(rate*1e-3*VDivDC)) //time per step in microseconds
         variable waittime = 0
         for(k=1; k<numsteps + 1; k+=1) //step voltage to target
            waittime = stopMSTimer(-2)
            SetA33220ADC(startV + stepV*k)
            do
            while((stopMSTimer(-2)-waittime)<steptime) //delay after each step to control sweep rate
         endfor
         SetA33220ADC(endV)
      endif
   catch
      Abort "Error or user abort in SweepVal"
   endtry
end

function do1d(idstr, start, stop, numdivs, rate)
   string idstr         // independent variable to sweep
   variable start       // starting value
   variable stop        // ending value
   variable numdivs     // number of points minus 1
   variable rate        // sweep rate per second

   rate = abs(rate) //make sure rate is positive
   variable actualrate = rate

   string searchstr
   variable startB = 0
   wave/T ExperimentalDetails
   Duplicate/O/R=[][0] ExperimentalDetails, ExperimentalDetailsName

   if(StringMatch(idstr[0,1], "Vg")) //parse Vg input by gate number
      searchstr = "Vg"
      variable numgates = strlen(idstr)
      variable y = 2
      string yokonum = "", checklist = ""
      for(y=2; y<numgates; y+=1)
         checklist += "Vg" + idstr[y] + ";"
      endfor
```

```
elseif(StringMatch(idstr, "B")) //check if heater on for 5 minutes
   searchstr = idstr
   SMSCleanBuffer()
   variable timeon = CheckSMSHeater()
   if(timeon==0)
      Print "SMS heater not on!  Unable to sweep B.  Aborting. (Do1D)"
      Abort "Do1D: SMS heater not on."
   elseif(timeon==-1)
      Print "SMS heater may not be on.  Make sure heater is on and retry.  Aborting. (Do1D)"
      Abort "Do1D: error from CheckSMSHeater."
   elseif(timeon<300) //heater must be on for at least 5 minutes
      printf "SMS heater turned on too recently.  "
      printf "Wait %d seconds and try again.  Aborting. (Do1D)\r", 300-timeon
      Abort "Do1D: SMS heater not on long enough."
   endif
   startB = getSMSField()
   if(StringMatch(num2str(startB), "NaN"))
      Print "Unable to determine current magnetic field.  Aborting. (Do1D)"
      Abort "Do1D: cannot get current field from GetSMSField."
   endif
else
   searchstr = idstr
endif

wave/T InVarWave
FindValue/TEXT=searchstr/TXOP=4 InVarWave //check if idstr is a valid independent variable
if((V_value == -1) || (V_value >= DimSize(InVarWave, 0)))
   printf "%s is not a valid independent variable (or is not included in InVarWave).  ", searchstr
   printf "Please try again. (Do1D)\r"
   Abort "Do1D: invalid independent variable."
else
   string ivarunits = InVarWave[V_value][1]
endif

variable maxval = 0, minval = 0, maxrate = 0
//checks inputs against Max, Min, and RateMax
if(StringMatch(searchstr, "Vg"))
   variable checklength = ItemsInList(checklist, ";")
   variable divider = Inf
   actualrate = rate * checklength //modify rate to account for sweeping multiple gates
   for(y=0; y<checklength; y+=1) //check each specified gate
      yokonum = StringFromList(y, checklist, ";")

      //make sure rate does not exceed RateMax
      FindValue/TEXT=(yokonum + "RateMax")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         maxrate = str2num(ExperimentalDetails[V_value][1])
         if(actualrate>maxrate)
            actualrate = maxrate
            printf "Actual rate too fast: "
            printf "reducing actual rate to %g %s. (Do1D)\r", maxrate, ivarunits + "/sec"
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "RateMax"
         printf "Sweep rate may be too fast. (Do1D)\r"
      endif

      FindValue/TEXT=(yokonum + "Divider")/TXOP=4 ExperimentalDetailsName //get divider value
      if(V_Value > -1)
         if(str2num(ExperimentalDetails[V_value][1]) > 0)
            divider = str2num(ExperimentalDetails[V_value][1])
         else
            printf "\"%s\" value not positive.  ", yokonum + "Divider"
```

```
            printf "Aborting. (Do1D)\r"
            Abort "Do1D: Vg#Divider not positive."
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  Aborting. (Do1D)\r", yokonum + "Divider"
         Abort "Do1D: cannot find Vg#Divider in ExperimentalDetails."
      endif

      //check hardware (Yoko) maximum
      FindValue/TEXT=(yokonum + "HardMax")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         variable hardmax = str2num(ExperimentalDetails[V_value][1])
         if(((start*divider)>hardmax) || ((stop*divider)>hardmax))
            printf "Start or stop value too high.  "
            printf "Yoko limited to %g V.  Aborting. (Do1D)\r", hardmax
            Abort "Do1D: start or stop value exceeds hardware maximum."
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "HardMax"
         printf "Unable to check if value exceeds maximum. (Do1D)\r"
      endif
      //check experimental (device) maximum
      FindValue/TEXT=(yokonum + "ExpMax")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         variable expmax = str2num(ExperimentalDetails[V_value][1])
         if((start>expmax) || (stop>expmax))
            printf "Start or stop value too high.  "
            printf "Limited to %g %s.  Aborting. (Do1D)\r", expmax, ivarunits
            Abort "Do1D: start or stop value exceeds maximum."
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "ExpMax"
         printf "Unable to check if value exceeds maximum. (Do1D)\r"
      endif

      //check hardware (Yoko) minimum
      FindValue/TEXT=(yokonum + "HardMin")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         variable hardmin = str2num(ExperimentalDetails[V_value][1])
         if(((start*divider)<hardmin) || ((stop*divider)<hardmin))
            printf "Start or stop value too low.  "
            printf "Yoko limited to %g V.  Aborting. (Do1D)\r", hardmin
            Abort "Do1D: start or stop value exceeds hardware minimum."
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "HardMin"
         printf "Unable to check if value exceeds minimum. (Do1D)\r"
      endif
      //check experimental (device) minimum
      FindValue/TEXT=(yokonum + "ExpMin")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         variable expmin = str2num(ExperimentalDetails[V_value][1])
         if((start<expmin) || (stop<expmin))
            printf "Start or stop value too low.  "
            printf "Limited to %g %s.  Aborting. (Do1D)\r", expmin, ivarunits
            Abort "Do1D: start or stop value exceeds  minimum."
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "ExpMin"
         printf "Unable to check if value exceeds minimum. (Do1D)\r"
      endif
   endfor
elseif(StringMatch(searchstr, "Idc"))
```

```
    FindValue/TEXT=("VtoI R")/TXOP=4 ExperimentalDetailsName //get Vdc to Idc conversion value
    if(V_Value > -1)
       variable VtoIR = str2num(ExperimentalDetails[V_value][1])
    else
       Print "\"VtoI R\" not found in ExperimentalDetails.  Aborting. (Do1D)"
       Abort "Do1D: VtoI R not found."
    endif
    FindValue/TEXT=(searchstr + "Max")/TXOP=4 ExperimentalDetailsName
    if(V_Value > -1)
       maxval = str2num(ExperimentalDetails[V_value][1])
       if(((start*1e-9*VtoIR)>maxval) || ((stop*1e-9*VtoIR)>maxval))
          printf "Start or stop value too high.  "
          printf "33220A limited to %g V.  Aborting. (Do1D)\r", maxval
          Abort "Do1D: start or stop value exceeds hardware maximum."
       endif
    else
       printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Max"
       printf "Unable to check if value exceeds maximum. (Do1D)\r"
    endif
    FindValue/TEXT=(searchstr + "Min")/TXOP=4 ExperimentalDetailsName
    if(V_Value > -1)
       minval = str2num(ExperimentalDetails[V_value][1])
       if(((start*1e-9*VtoIR)<minval) || ((start*1e-9*VtoIR)<minval))
          printf "Start or stop value too low.  "
          printf "33220A limited to %g V.  Aborting. (Do1D)\r", minval
          Abort "Do1D: start or stop value exceeds hardware minimum."
       endif
    else
       printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Min"
       printf "Unable to check if value exceeds minimum. (Do1D)\r"
    endif
    FindValue/TEXT=(searchstr + "RateMax")/TXOP=4 ExperimentalDetailsName
    if(V_Value > -1)
       maxrate = str2num(ExperimentalDetails[V_value][1])
       if(rate>maxrate)
          rate = maxrate
          printf "Rate too fast: reducing rate to %g %s. (Do1D)\r", maxrate, ivarunits + "/sec"
       endif
    else
       printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "RateMax"
       printf "Sweep rate may be too fast. (Do1D)\r"
    endif
 elseif(StringMatch(searchstr, "Vdc"))
    FindValue/TEXT=("VdcDivider")/TXOP=4 ExperimentalDetailsName //get DC voltage divider
    if(V_Value > -1)
       variable VDivDC = str2num(ExperimentalDetails[V_value][1])
    else
       Print "\"VdcDivider\" not found in ExperimentalDetails.  Aborting. (Do1D)"
       Abort "Do1D: VdcDivider not found."
    endif
    FindValue/TEXT=(searchstr + "Max")/TXOP=4 ExperimentalDetailsName
    if(V_Value > -1)
       maxval = str2num(ExperimentalDetails[V_value][1])
       if(((start*1e-3*VDivDC)>maxval) || ((stop*1e-3*VDivDC)>maxval))
          printf "Start or stop value too high.  "
          printf "33220A limited to %g V.  Aborting. (Do1D)\r", maxval
          Abort "Do1D: start or stop value exceeds hardware maximum."
       endif
    else
       printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Max"
       printf "Unable to check if value exceeds maximum. (Do1D)\r"
    endif
    FindValue/TEXT=(searchstr + "Min")/TXOP=4 ExperimentalDetailsName
```

```
    if(V_Value > -1)
       minval = str2num(ExperimentalDetails[V_value][1])
       if(((start*1e-3*VDivDC)<minval) || ((start*1e-3*VDivDC)<minval))
          printf "Start or stop value too low.  "
          printf "33220A limited to %g V.  Aborting. (Do1D)\r", minval
          Abort "Do1D: start or stop value exceeds hardware minimum."
       endif
    else
       printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Min"
       printf "Unable to check if value exceeds minimum. (Do1D)\r"
    endif
    FindValue/TEXT=(searchstr + "RateMax")/TXOP=4 ExperimentalDetailsName
    if(V_Value > -1)
       maxrate = str2num(ExperimentalDetails[V_value][1])
       if(rate>maxrate)
          rate = maxrate
          printf "Rate too fast: reducing rate to %g %s. (Do1D)\r", maxrate, ivarunits + "/sec"
       endif
    else
       printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "RateMax"
       printf "Sweep rate may be too fast. (Do1D)\r"
    endif
 else
    FindValue/TEXT=(searchstr + "Max")/TXOP=4 ExperimentalDetailsName
    if(V_Value > -1)
       maxval = str2num(ExperimentalDetails[V_value][1])
       if((start>maxval) || (stop>maxval))
          printf "Start or stop value too high.  "
          printf "Limited to %g %s.  Aborting. (Do1D)\r", maxval, ivarunits
          Abort "Do1D: start or stop value exceeds maximum."
       endif
    else
       printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Max"
       printf "Unable to check if value exceeds maximum. (Do1D)\r"
    endif
    FindValue/TEXT=(searchstr + "Min")/TXOP=4 ExperimentalDetailsName
    if(V_Value > -1)
       minval = str2num(ExperimentalDetails[V_value][1])
       if((start<minval) || (stop<minval))
          printf "Start or stop value too low.  "
          printf "Limited to %g %s.  Aborting. (Do1D)\r", minval, ivarunits
          Abort "Do1D: start or stop value exceeds minimum."
       endif
    else
       printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Min"
       printf "Unable to check if value exceeds minimum. (Do1D)\r"
    endif
    FindValue/TEXT=(searchstr + "RateMax")/TXOP=4 ExperimentalDetailsName
    if(V_Value > -1)
       maxrate = str2num(ExperimentalDetails[V_value][1])
       if(rate>maxrate)
          rate = maxrate
          printf "Rate too fast: reducing rate to %g %s. (Do1D)\r", maxrate, ivarunits + "/sec"
       endif
    else
       printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "RateMax"
       printf "Sweep rate may be too fast. (Do1D)\r"
    endif
 endif

 //after this point the experiment changes, and will not revert after an abort

 string recordstr = "", recordstr2 = ""
```

```
sprintf recordstr, "%s: %s: Do1D(%s,", date(), time(), idstr
sprintf recordstr2, "%g, %g, %g, %g)\r", start, stop, numdivs, rate
recordstr += recordstr2
Notebook Record, selection={StartofFile, StartofFile}, text = recordstr
SaveNotebook/O/P=savepath/S=6 Record as "Record.txt"

FindValue/TEXT=("Independent")/TXOP=4 ExperimentalDetailsName
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = idstr
else
   print "\"Independent\" not found in ExperimentalDetails."
   print "Unable to record independent variable. (Do1D)"
endif

NVAR KeepGraphsFlag
string graphlist = WinList("Graph*", ";", "WIN:1") //get list of unnamed open graphs
variable numgraphs = ItemsInList(graphlist, ";"), k = 0
if(KeepGraphsFlag == 0)
   for(k = 0; k<numgraphs; k+=1)
      KillWindow $StringFromList(k, graphlist, ";") //kill all unnamed open graphs
   endfor
   string killlist = DataWaveList() //list of all data waves (with name matching proper format)
   variable killlength = ItemsInList(killlist, ";")
   //kill all data waves not in use
   for(k=0; k<killlength; k+=1)
      //does not kill waves which are in graphs, tables, etc.
      KillWaves/Z $StringFromList(k, killlist, ";")
   endfor
elseif(KeepGraphsFlag == 1)
   string wlist = "" //list of waves in graph
   string commandstr = ""
   variable numwaves = 0 //number of waves in graph
   variable waveappended = 0 //indicates if new wave could be appended to existing graph
else
   printf "KeepGraphsFlag value of %g invalid.  ", KeepGraphsFlag
   printf "Must be 0 or 1.  Aborting. (Do1D)\r"
   Abort "Do1D: invalid value of KeepGraphsFlag"
endif

variable numpts=numdivs+1
//update number of x and y points in ExperimentalDetails
FindValue/TEXT=("NumPoints X")/TXOP=4 ExperimentalDetailsName
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = num2istr(numpts)
else
   Print "\"NumPoints X\" not found in ExperimentalDetails."
   Print "Unable to record number of x-dimension points. (Do1D)"
endif
FindValue/TEXT=("NumPoints Y")/TXOP=4 ExperimentalDetailsName
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = "0"
else
   Print "\"NumPoints Y\" not found in ExperimentalDetails."
   Print "Unable to record number of y-dimension points. (Do1D)"
endif

variable nextwaveindex=nextwave(increment = 1) //increment data index by 1
wave/T datatypedmm
variable maxmeasurements = DimSize(datatypedmm, 0)
make/O/T/N=(maxmeasurements) wavenamelist = ""
string currentwavename = ""
variable starttime = 0, datatime = 0
variable n = 0, m = 0
```

```
make/O/N=(maxmeasurements, 2) do1dScaleFactors //wave to keep track of offsets and scale factors
variable scale, offset

string do1dDeVars = "", recordvalue = "", dmmlipa = "", currentdatatype = ""
//update Start, Stop, and Rate values in ExperimentalDetails
if(StringMatch(searchstr, "Vg"))
   for(y=0; y<checklength; y+=1)
      yokonum = StringFromList(y, checklist, ";")
      FindValue/TEXT=(yokonum+"Start")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         sprintf recordvalue, "%g", start
         ExperimentalDetails[V_value][1] = recordvalue
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "Start"
         printf "Unable to record start value. (Do1D)\r"
      endif
      FindValue/TEXT=(yokonum+"Stop")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         sprintf recordvalue, "%g", stop
         ExperimentalDetails[V_value][1] = recordvalue
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "Stop"
         printf "Unable to record stop value. (Do1D)\r"
      endif
      FindValue/TEXT=(yokonum+"Rate")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         sprintf recordvalue, "%g", actualrate
         ExperimentalDetails[V_value][1] = recordvalue
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "Rate"
         printf "Unable to record sweep rate. (Do1D)\r"
      endif
   endfor
else
   FindValue/TEXT=(searchstr+"Start")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      sprintf recordvalue, "%g", start
      ExperimentalDetails[V_value][1] = recordvalue
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Start"
      printf "Unable to record start value. (Do1D)\r"
   endif
   FindValue/TEXT=(searchstr+"Stop")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      sprintf recordvalue, "%g", stop
      ExperimentalDetails[V_value][1] = recordvalue
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Stop"
      printf "Unable to record stop value. (Do1D)\r"
   endif
   FindValue/TEXT=(searchstr+"Rate")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      sprintf recordvalue, "%g", rate
      ExperimentalDetails[V_value][1] = recordvalue
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr + "Rate"
      printf "Unable to record sweep rate. (Do1D)\r"
   endif
endif

variable windowleft, windowtop, windowright, windowbottom
for(n=0; n < maxmeasurements; n+=1) //step though each dependent variable to be measured
```

```
currentdatatype = datatypedmm[n][0]
if(!stringmatch(currentdatatype,"-"))
   scale = 1 //for ReadDMM returning V
   offset = 0

   for(m=0; m<maxmeasurements; m+=1)
      if((m != n) && (stringmatch(currentdatatype, datatypedmm[m][0])))
         printf "Dependent variable %s chosen more than once ", currentdatatype
         printf "and will be overwritten.  Aborting. (Do1D)\r"
         Abort "Do1D: dependent variable chosen multiple times."
      endif
   endfor

   //get dmm offset (y-intercept)
   FindValue/TEXT=("DMMoffset" + datatypedmm[n][2])/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      offset += str2num(ExperimentalDetails[V_value][1])
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", "DMMoffset" + datatypedmm[n][2]
      printf "Offset not changed. (Do1D)\r"
   endif
   //get dmm scale factor
   FindValue/TEXT=("DMMscale" + datatypedmm[n][2])/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      scale *= str2num(ExperimentalDetails[V_Value][1])
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", "DMMscale" + datatypedmm[n][2]
      printf "Scale not changed. (Do1D)\r"
   endif
   if(!stringmatch(datatypedmm[n][4], "0"))
      //get lock-in sensitivity
      FindValue/TEXT=("LIsens" + datatypedmm[n][4])/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         scale *= str2num(ExperimentalDetails[V_value][1])/10
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", "LIsens" + datatypedmm[n][3]
         printf "Scale not changed. (Do1D)\r"
      endif
   endif
   if(!stringmatch(datatypedmm[n][5], "0"))
      //get pre-amp amplification
      FindValue/TEXT=("PAamp" + datatypedmm[n][5])/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         scale /= str2num(ExperimentalDetails[V_value][1])
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", "PAamp" + datatypedmm[n][4]
         printf "Scale not changed. (Do1D)\r"
      endif
   endif

   scale *= ScaleFactor(n)

   //update do1dScaleFactors wave
   do1dScaleFactors[n][0] = offset
   do1dScaleFactors[n][1] = scale

   do1dDeVars += currentdatatype+ ";" //keep track of dependent variables to be measured
   //record DMM, lock-in, preamp chain
   dmmlipa += datatypedmm[n][2] + datatypedmm[n][4] + datatypedmm[n][5] + ";"

   sprintf currentwavename, "%s_%s_%d", datatypedmm[n][0], idstr, nextwaveindex
   wavenamelist[n] = currentwavename //keep list of wave names
   make/O/N=(numpts) $currentwavename = NaN //make the wave
```

```
      SetScale/I x start, stop, $currentwavename //set x scaling
      if(KeepGraphsFlag == 0) //make new graphs for next set of data
         Display $currentwavename //create graph
         Label bottom, idstr + " (" + ivarunits + ")" //label x-axis
         Label left, datatypedmm[n][0] + " (" + datatypedmm[n][1] +")" //label y-axis
         GraphStyle() //edit function to change graph style
         //move graph window
         movewindow/I (mod(n, 3)*3), (floor(n/3)*3), (mod(n, 3)*3 + 2.95), (floor(n/3)*3 + 2.2)
      elseif(KeepGraphsFlag == 1) //display data on old graphs
         waveappended = 0
         for(m=0; m<numgraphs; m+=1) //check each graph for match of devar
            //writes list of waves in graph to W_WaveList
            commandstr = "GetWindow " + StringFromList(m, graphlist, ";") + ", wavelist"
            Execute/Q commandstr
            wlist = TWave2Str(W_WaveList) //generate wlist
            numwaves = ItemsInList(wlist, ";") //number of waves on graph
            for(k=0; k<numwaves; k+=1) //check each wave to see if match devar
               //if match, append new trace
               if(stringmatch(StringFromList(k, wlist, ";"), currentdatatype + "_*"))
                  commandstr = "AppendToGraph/W=" + StringFromList(m, graphlist, ";")
                  commandstr += " " + currentwavename
                  Execute/Q commandstr
                  waveappended = 1
                  break
               endif
            endfor
         endfor
         if(waveappended == 0)
            Display $currentwavename //create graph
            Label bottom, idstr + " (" + ivarunits + ")" //label x-axis
            Label left, datatypedmm[n][0] + " (" + datatypedmm[n][1] +")" //label y-axis
            GraphStyle() //edit function to change graph style
            //move graph window
            windowleft = (mod(n+numgraphs, 3)*3)
            windowtop = (floor((n+numgraphs)/3)*3)
            windowright = (mod(n+numgraphs, 3)*3 + 2.95)
            windowbottom = (floor((n+numgraphs)/3)*3 + 2.2)
            movewindow/I windowleft, windowtop, windowright, windowbottom
         endif
      endif
   endif
endfor
//update dependent variable in ExperimentalDetails
FindValue/TEXT=("Dependent")/TXOP=4 ExperimentalDetailsName
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = do1dDeVars
else
   Print "\"Dependent\" not found in ExperimentalDetails."
   Print "Unable to record dependent variables. (Do1D)"
endif
//update dmm/lock-in/preamp info in ExperimentalDetails
FindValue/TEXT=("DMMLIPA")/TXOP=4 ExperimentalDetailsName
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = dmmlipa
else
   Print "\"DMMLIPA\" not found in ExperimentalDetails."
   Print "Unable to record dependent variables. (Do1D)"
endif

FindValue/TEXT=("Date")/TXOP=4 ExperimentalDetailsName //update date
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = date()
else
```

```
   Print "\"Date\" not found in ExperimentalDetails.  Unable to record date. (Do1D)"
endif
FindValue/TEXT=("StartTime")/TXOP=4 ExperimentalDetailsName //update start time
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = time()
else
   Print "\"StartTime\" not found in ExperimentalDetails.  Unable to record time. (Do1D)"
endif

//save ExperimentalDetails to file
Duplicate/O/T/R=[][1] ExperimentalDetails do1dDetails
ExperimentalDetails[][DimSize(ExperimentalDetails, 1)-1] = do1dDetails[p]
Concatenate/T {ExperimentalDetailsName}, ExperimentalDetails
SaveDetails()

//constants for mixing chamber temperature fit
variable A0 = 411.7925271369486, A1 = -487.3966692536534, A2 = 233.1118681633504
variable A3 = -55.85454979733308, A4 = 6.690672513604219, A5 = -0.3205752792140923

make/O/N=(numpts) do1dInVals //wave of independent variable points
do1dInVals=start+p*(stop-start)/numdivs

// data taking loop
make/O/N=(maxmeasurements) do1dcurrentvals = NaN
make/O do1dcurrentwave
SweepVal(idstr, do1dInVals[0], actualrate, startB)
//wait(1)
try
   starttime = stopMSTimer(-2)
   for(m=0; m<numpts; m+=1)
      SweepVal(idstr, do1dInVals[m], actualrate, do1dInVals[m-1]) //set independent variable
      TakeData(do1dcurrentvals) //record data from dmms
      for(n=0; n<maxmeasurements; n+=1) //step through waves being taken
         if(!stringmatch(datatypedmm[n][0], "-"))
            currentwavename = wavenamelist[n]
            wave do1dcurrentwave = $currentwavename
            //update wave with with scaled data
            do1dcurrentwave[m] = (do1dcurrentvals[n]-do1dScaleFactors[n][0])*do1dScaleFactors[n][1]
            if(stringmatch(idstr, "Time")) //scale x-axis with time
               datatime = stopMSTimer(-2)
               SetScale/P x, 0, ((datatime-starttime)*1e-6/(m+1)), $currentwavename
            endif
            if(stringmatch(datatypedmm[n][0], "Tmc")) //implement nonlinear temperature fit
               do1dcurrentwave[m] = A0 + A1 * log(do1dcurrentwave[m])
               do1dcurrentwave[m] += A2*(log(do1dcurrentwave[m]))^2
               do1dcurrentwave[m] += A3*(log(do1dcurrentwave[m]))^3
               do1dcurrentwave[m] += A4*(log(do1dcurrentwave[m]))^4
               do1dcurrentwave[m] += A5*(log(do1dcurrentwave[m]))^5
               do1dcurrentwave[m] = 10^do1dcurrentwave[m]
            endif
         endif
      endfor
      DoUpdate //Comment this line if no data will be displayed during acquisition JBM060828
   endfor
   datatime = stopMSTimer(-2)
catch
   datatime = stopMSTimer(-2)
   DoUpdate
endtry
AllDMMLocal() //do we want this?
try //save waves and update ExperimentalDetails even if function aborted
   for(n=0; n<maxmeasurements; n+=1)
      if(!stringmatch(datatypedmm[n][0], "-")) //save waves as Igor binary files
```

```
            currentwavename = wavenamelist[n]
            Save/P=binarypath $currentwavename
            Save/P=backuppath $currentwavename
         endif
      endfor

      //update Start, Stop, and Rate to current values (may not match input if function was aborted)
      if(StringMatch(searchstr, "Vg"))
         for(y=0; y<checklength; y+=1)
            yokonum = StringFromList(y, checklist, ";")
            FindValue/TEXT=(yokonum+"Start")/TXOP=4 ExperimentalDetailsName
            if(V_Value > -1)
               sprintf recordvalue, "%g", do1dInVals[m]
               ExperimentalDetails[V_value][1] = recordvalue
            else
               printf "Warning: unable to update %s after sweep. (Do1D)\r", yokonum + "Start"
            endif
            FindValue/TEXT=(yokonum+"Stop")/TXOP=4 ExperimentalDetailsName
            if(V_Value > -1)
               sprintf recordvalue, "%g", do1dInVals[m]
               ExperimentalDetails[V_value][1] = recordvalue
            else
               printf "Warning: unable to update %s after sweep. (Do1D)\r", yokonum + "Stop"
            endif
            FindValue/TEXT=(yokonum+"Rate")/TXOP=4 ExperimentalDetailsName
            if(V_Value > -1)
               ExperimentalDetails[V_value][1] = "0"
            else
               printf "Warning: unable to update %s after sweep. (Do1D)\r", yokonum + "Rate"
            endif
         endfor
      else
         FindValue/TEXT=(searchstr+"Start")/TXOP=4 ExperimentalDetailsName
         if(V_Value > -1)
            sprintf recordvalue, "%g", do1dInVals[m]
            ExperimentalDetails[V_value][1] = recordvalue
         else
            printf "Warning: unable to update %s after sweep. (Do1D)\r", searchstr + "Start"
         endif
         FindValue/TEXT=(searchstr+"Stop")/TXOP=4 ExperimentalDetailsName
         if(V_Value > -1)
            sprintf recordvalue, "%g", do1dInVals[m]
            ExperimentalDetails[V_value][1] = recordvalue
         else
            printf "Warning: unable to update %s after sweep. (Do1D)\r", searchstr + "Stop"
         endif
         FindValue/TEXT=(searchstr+"Rate")/TXOP=4 ExperimentalDetailsName
         if(V_Value > -1)
            ExperimentalDetails[V_value][1] = "0"
         else
            printf "Warning: unable to update %s after sweep. (Do1D)\r", searchstr + "Rate"
         endif
      endif
   catch
      Print "Warning: function aborted, data may not be saved."
      Print "Check that all files are properly saved. (Do1D)"
   endtry
   printf "Created waves #%d, finished at %s, ",nextwaveindex,time()
   printf "elapsed time %6.3f min.\r",((datatime-starttime)*1e-6/60)
   sprintf recordstr, "Created waves #%d, finished at %s, ",nextwaveindex,time()
   sprintf recordstr2, "elapsed time %6.3f min.\r",((datatime-starttime)*1e-6/60)
   recordstr += recordstr2
   Notebook Record, selection={StartofFile, StartofFile}, text = recordstr
```

```
   SaveNotebook/O/P=savepath/S=6 Record as "Record.txt"
end

function do2d(idstr1, srt1, stop1, ndiv1, rate1, idstr2, srt2, stop2, ndiv2, rate2, [bkrate, delay])
   //1: fast axis, y-axis; 2: slow axis, x-axis
   string idstr1, idstr2
   variable srt1, srt2        // starting value
   variable stop1, stop2         // ending value
   variable ndiv1, ndiv2      // number of points minus 1
   variable rate1, rate2         // sweep rate per second
   variable bkrate //rate to sweep fast axis back to srt1
   variable delay //seconds to wait before each fast axis sweep

   if(StringMatch(idstr1, idstr2))
      Print "Both independent variables are the same.  Cannot sweep.  Aborting (Do2D)"
      Abort "Do2D: same independent variables."
   endif

   //make sure rates are positive)
   rate1 = abs(rate1)
   rate2 = abs(rate2)
   bkrate = abs(bkrate)
   variable actualrate1 = rate1
   variable actualrate2 = rate2
   variable actualbkrate = bkrate

   string searchstr1, searchstr2
   variable startB = 0
   wave/T ExperimentalDetails
   Duplicate/O/R=[][0] ExperimentalDetails, ExperimentalDetailsName

   if(ParamIsDefault(delay))
      delay = 0 //default delay before each fast axis sweep
   endif

   variable numgates = 0, y = 0
   string yokonum = ""
   //parse gate numbers
   if(StringMatch(idstr1[0,1], "Vg"))
      searchstr1 = "Vg"
      numgates = strlen(idstr1)
      string checklist1 = ""
      for(y=2; y<numgates; y+=1)
         checklist1 += "Vg" + idstr1[y] + ";"
      endfor
   else
      searchstr1 = idstr1
   endif
   if(StringMatch(idstr2[0,1], "Vg"))
      searchstr2 = "Vg"
      numgates = strlen(idstr2)
      string checklist2 = ""
      for(y=2; y<numgates; y+=1)
         checklist2 += "Vg" + idstr2[y] + ";"
      endfor
   else
      searchstr2 = idstr2
   endif


   wave/T InVarWave
   //check that idstr1 and idstr2 are valid independent variables
   FindValue/TEXT=searchstr1/TXOP=4 InVarWave
```

```
if((V_value == -1) || (V_value >= DimSize(InVarWave, 0)))
   printf "%s is not a valid independent variable ", searchstr1
   printf "(or is not included in InVarWave).  Please try again. (Do2D)\r"
   Abort "Do2D: invalid independent variable 1."
else
   string ivarunits1 = InVarWave[V_value][1]
endif
FindValue/TEXT=searchstr2/TXOP=4 InVarWave
if((V_value == -1) || (V_value >= DimSize(InVarWave, 0)))
   printf "%s is not a valid independent variable ", searchstr2
   printf "(or is not included in InVarWave).  Please try again. (Do2D)\r"
   Abort "Do2D: invalid independent variable 2."
else
   string ivarunits2 = InVarWave[V_value][1]
endif

//check that heater on for at least 5 minutes
if(StringMatch(searchstr1, "B") || StringMatch(searchstr2, "B"))
   SMSCleanBuffer()
   variable timeon = CheckSMSHeater()
   if(timeon==0)
      Print "SMS heater not on!  Unable to sweep B.  Aborting. (Do2D)"
      Abort "Do2D: SMS heater not on."
   elseif(timeon==-1)
      Print "SMS heater may not be on.  Make sure heater is on and retry.  Aborting. (Do2D)"
      Abort "Do2D: error from CheckSMSHeater."
   elseif(timeon<300) //heater must be on for at least 5 minutes
      printf "SMS heater turned on too recently.  "
      printf "Wait %d seconds and try again.  Aborting. (Do2D)\r", 300-timeon
      Abort "Do2D: SMS heater not on long enough."
   endif
   startB = GetSMSField()
   if(StringMatch(num2str(startB), "NaN"))
      Print "Unable to determine current magnetic field.  Aborting. (Do2D)"
      Abort "Do2D: cannot get current field from GetSMSField."
   endif
endif

variable maxval1 = 0, maxval2 = 0, minval1 = 0, minval2 = 0
variable maxrate1 = 0, maxrate2 = 0, divider = Inf
variable VtoIR = Inf, VDivDC = Inf
//check inputs against Max, Min, and RateMax values
if(StringMatch(searchstr1, "Vg"))
   variable checklength1 = ItemsInList(checklist1, ";")
   actualrate1 = rate1 * checklength1 //account for sweeping multiple gates
   actualbkrate = bkrate * checklength1
   for(y=0; y<checklength1; y+=1)
      yokonum = StringFromList(y, checklist1, ";")

      FindValue/TEXT=(yokonum + "Divider")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         if(str2num(ExperimentalDetails[V_value][1]) > 0)
            divider = str2num(ExperimentalDetails[V_value][1])
         else
            printf "\"%s\" value not positive.  Aborting. (Do2D)\r", yokonum + "Divider"
            Abort "Do2D: Vg#Divider not positive."
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "Divider"
         printf "Aborting. (Do2D)\r"
         Abort "Do2D: cannot find Vg#Divider in ExperimentalDetails."
      endif
```

```
FindValue/TEXT=(yokonum + "HardMax")/TXOP=4 ExperimentalDetailsName
if(V_Value > -1)
   variable hardmax1 = str2num(ExperimentalDetails[V_value][1])
   if(((srt1*divider)>hardmax1) || ((stop1*divider)>hardmax1))
      printf "Srt1 or stop1 value too high.  "
      printf "Yoko limited to %g V.  Aborting. (Do2D)\r", hardmax1
      Abort "Do2D: srt1 or stop1 value exceeds hardware maximum."
   endif
else
   printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "HardMax"
   printf "Unable to check if value exceeds maximum. (Do2D)\r"
endif
FindValue/TEXT=(yokonum + "ExpMax")/TXOP=4 ExperimentalDetailsName
if(V_Value > -1)
   variable expmax1 = str2num(ExperimentalDetails[V_value][1])
   if((srt1>expmax1) || (stop1>expmax1))
      printf "Srt1 or stop1 value too high.  "
      printf "Limited to %g %s.  Aborting. (Do2D)\r", expmax1, ivarunits1
      Abort "Do2D: srt1 or stop1 value exceeds maximum."
   endif
else
   printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "ExpMax"
   printf "Unable to check if value exceeds maximum. (Do2D)\r"
endif

FindValue/TEXT=(yokonum + "HardMin")/TXOP=4 ExperimentalDetailsName
if(V_Value > -1)
   variable hardmin1 = str2num(ExperimentalDetails[V_value][1])
   if(((srt1*divider)<hardmin1) || ((stop1*divider)<hardmin1))
      printf "Srt1 or stop1 value too low.  "
      printf "Yoko limited to %g V.  Aborting. (Do2D)\r", hardmin1
      Abort "Do2D: srt1 or stop1 value exceeds hardware minimum."
   endif
else
   printf "\"%s\" not found in ExperimentalDetails.  ", yokonum+ "HardMin"
   printf "Unable to check if value exceeds minimum. (Do2D)\r"
endif
FindValue/TEXT=(yokonum + "ExpMin")/TXOP=4 ExperimentalDetailsName
if(V_Value > -1)
   variable expmin1 = str2num(ExperimentalDetails[V_value][1])
   if((srt1<expmin1) || (stop1<expmin1))
      printf "Srt1 or stop1 value too low.  "
      printf "Limited to %g %s.  Aborting. (Do2D)\r", expmin1, ivarunits1
      Abort "Do2D: srt1 or stop1 value exceeds minimum."
   endif
else
   printf "\"%s\" not found in ExperimentalDetails.  ", yokonum+ "ExpMin"
   printf "Unable to check if value exceeds minimum. (Do2D)\r"
endif

FindValue/TEXT=(yokonum + "RateMax")/TXOP=4 ExperimentalDetailsName
if(V_Value > -1)
   maxrate1 = str2num(ExperimentalDetails[V_value][1])
   if(actualrate1>maxrate1)
      actualrate1 = maxrate1
      printf "Actualrate1 too fast: "
      printf "reducing actual rate to %g %s. (Do2D)\r", maxrate1, ivarunits1 + "/sec."
   endif
   if(!ParamIsDefault(bkrate))
      if(actualbkrate>maxrate1)
         actualbkrate = maxrate1
         printf "Actualbkrate too fast: "
```

```
            printf "reducing rate to %g %s. (Do2D)\r", maxrate1, ivarunits1 + "/sec."
         endif
         if(actualbkrate == 0)
            actualbkrate = actualrate1
            Print "Bkrate specified as 0.  Setting equal to fast axis rate."
         endif
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr1 + "RateMax"
      printf "Sweep rate may be too fast. (Do2D)\r"
   endif
endfor
elseif(StringMatch(searchstr1, "Idc"))
   FindValue/TEXT=("VtoI R")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      VtoIR = str2num(ExperimentalDetails[V_value][1])
   else
      Print "\"VtoI R\" not found in ExperimentalDetails.  Aborting. (Do2D)"
      Abort "Do2D: VtoI R not found."
   endif
   FindValue/TEXT=(searchstr1 + "Max")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      maxval1 = str2num(ExperimentalDetails[V_value][1])
      if(((srt1*1e-9*VtoIR)>maxval1) || ((srt1*1e-9*VtoIR)>maxval1))
         printf "Srt1 or stop1 value too high.  "
         printf "33220A limited to %g (srt1*1e-9*VtoIR).  Aborting. (Do2D)\r", maxval1
         Abort "Do2D: srt1 or stop1 value exceeds hardware maximum."
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr1 + "Max"
      printf "Unable to check if value exceeds maximum. (Do2D)\r"
   endif
   FindValue/TEXT=(searchstr1 + "Min")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      minval1 = str2num(ExperimentalDetails[V_value][1])
      if(((srt1*1e-9*VtoIR)<minval1) || ((srt1*1e-9*VtoIR)<minval1))
         printf "Srt1 or stop1 value too low.  "
         printf "33220A limited to %g V.  Aborting. (Do2D)\r", minval1
         Abort "Do2D: srt1 or stop1 value exceeds hardware minimum."
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr1+ "Min"
      printf "Unable to check if value exceeds minimum. (Do2D)\r"
   endif
   FindValue/TEXT=(searchstr1 + "RateMax")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      maxrate1 = str2num(ExperimentalDetails[V_value][1])
      if(rate1>maxrate1)
         rate1 = maxrate1
         printf "Rate1 too fast: reducing rate to %g %s. (Do2D)\r", maxrate1, ivarunits1 + "/sec."
      endif
      if(!ParamIsDefault(bkrate))
         if(bkrate>maxrate1)
            bkrate = maxrate1
            printf "Bkrate too fast: "
            printf "reducing rate to %g %s. (Do2D)\r", maxrate1, ivarunits1 + "/sec."
         endif
         if(bkrate == 0)
            bkrate = rate1
            Print "Bkrate specified as 0.  Setting equal to fast axis rate."
         endif
      endif
   else
```

```
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr1 + "RateMax"
      printf "Sweep rate may be too fast. (Do2D)\r"
   endif
elseif(StringMatch(searchstr1, "Vdc"))
   FindValue/TEXT=("VdcDivider")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      VdivDC = str2num(ExperimentalDetails[V_value][1])
   else
      Print "\"VdcDivider\" not found in ExperimentalDetails.  Aborting. (Do2D)"
      Abort "Do2D: VdcDivider not found."
   endif
   FindValue/TEXT=(searchstr1 + "Max")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      maxval1 = str2num(ExperimentalDetails[V_value][1])
      if(((srt1*1e-3*VDivDC)>maxval1) || ((srt1*1e-3*VDivDC)>maxval1))
         printf "Srt1 or stop1 value too high.  "
         printf "33220A limited to %g (srt1*1e-9*VtoIR).  Aborting. (Do2D)\r", maxval1
         Abort "Do2D: srt1 or stop1 value exceeds hardware maximum."
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr1 + "Max"
      printf "Unable to check if value exceeds maximum. (Do2D)\r"
   endif
   FindValue/TEXT=(searchstr1 + "Min")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      minval1 = str2num(ExperimentalDetails[V_value][1])
      if(((srt1*1e-3*VDivDC)<minval1) || ((srt1*1e-3*VDivDC)<minval1))
         printf "Srt1 or stop1 value too low.  "
         printf "33220A limited to %g V.  Aborting. (Do2D)\r", minval1
         Abort "Do2D: srt1 or stop1 value exceeds hardware minimum."
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr1+ "Min"
      printf "Unable to check if value exceeds minimum. (Do2D)\r"
   endif
   FindValue/TEXT=(searchstr1 + "RateMax")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      maxrate1 = str2num(ExperimentalDetails[V_value][1])
      if(rate1>maxrate1)
         rate1 = maxrate1
         printf "Rate1 too fast: reducing rate to %g %s. (Do2D)\r", maxrate1, ivarunits1 + "/sec."
      endif
      if(!ParamIsDefault(bkrate))
         if(bkrate>maxrate1)
            bkrate = maxrate1
            printf "Bkrate too fast: "
            printf "reducing rate to %g %s. (Do2D)\r", maxrate1, ivarunits1 + "/sec."
         endif
         if(bkrate == 0)
            bkrate = rate1
            Print "Bkrate specified as 0.  Setting equal to fast axis rate. (Do2D)"
         endif
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr1 + "RateMax"
      printf "Sweep rate may be too fast. (Do2D)\r"
   endif
else
   FindValue/TEXT=(searchstr1 + "Max")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      maxval1 = str2num(ExperimentalDetails[V_value][1])
      if((srt1>maxval1) || (stop1>maxval1))
         printf "Srt1 or stop1 value too high.  "
```

```
            printf "Limited to %g %s.  Aborting. (Do2D)\r", maxval1, ivarunits1
            Abort "Do2D: srt1 or stop1 value exceeds maximum."
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", searchstr1 + "Max"
         printf "Unable to check if value exceeds maximum. (Do2D)\r"
      endif
      FindValue/TEXT=(searchstr1 + "Min")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         minval1 = str2num(ExperimentalDetails[V_value][1])
         if((srt1<minval1) || (stop1<minval1))
            printf "Srt1 or stop1 value too low.  "
            printf "Limited to %g %s.  Aborting. (Do2D)\r", minval1, ivarunits1
            Abort "Do2D: srt1 or stop1 value exceeds minimum."
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", searchstr1+ "Min"
         printf "Unable to check if value exceeds minimum. (Do2D)\r"
      endif
      FindValue/TEXT=(searchstr1 + "RateMax")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         maxrate1 = str2num(ExperimentalDetails[V_value][1])
         if(rate1>maxrate1)
            rate1 = maxrate1
            printf "Rate1 too fast: reducing rate to %g %s. (Do2D)\r", maxrate1, ivarunits1 + "/sec."
         endif
         if(!ParamIsDefault(bkrate))
            if(bkrate>maxrate1)
               bkrate = maxrate1
               printf "Bkrate too fast: "
               printf "reducing rate to %g %s. (Do2D)\r", maxrate1, ivarunits1 + "/sec."
            endif
            if(bkrate == 0)
               bkrate = rate1
               Print "Bkrate specified as 0.  Setting equal to fast axis rate."
            endif
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", searchstr1 + "RateMax"
         printf "Sweep rate may be too fast. (Do2D)\r"
      endif
   endif

   if(StringMatch(searchstr2, "Vg"))
      variable checklength2 = ItemsInList(checklist2, ";")
      actualrate2 = rate2 * checklength2 //account for sweeping multiple gates
      for(y=0; y<checklength2; y+=1)
         yokonum = StringFromList(y, checklist2, ";")

         FindValue/TEXT=(yokonum + "Divider")/TXOP=4 ExperimentalDetailsName
         if(V_Value > -1)
            divider = str2num(ExperimentalDetails[V_value][1])
         else
            printf "\"%s\" not found in ExperimentalDetails.  Aborting. (Do2D)", yokonum + "Divider"
            Abort "Do2D: cannot find Vg#Divider in ExperimentalDetails."
         endif

         FindValue/TEXT=(yokonum + "HardMax")/TXOP=4 ExperimentalDetailsName
         if(V_Value > -1)
            variable hardmax2 = str2num(ExperimentalDetails[V_value][1])
            if(((srt2*divider)>hardmax2) || ((stop2*divider)>hardmax2))
               printf "Srt2 or stop2 value too high.  "
               printf "Yoko limited to %g V.  Aborting. (Do2D)\r", hardmax2
```

```
            Abort "Do2D: srt2 or stop2 value exceeds hardware maximum."
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "HardMax"
         printf "Unable to check if value exceeds maximum. (Do2D)\r"
      endif
      FindValue/TEXT=(yokonum + "ExpMax")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         variable expmax2 = str2num(ExperimentalDetails[V_value][1])
         if((srt2>expmax2) || (stop2>expmax2))
            printf "Srt2 or stop2 value too high.  "
            printf "Limited to %g %s.  Aborting. (Do2D)\r", expmax2, ivarunits2
            Abort "Do2D: srt2 or stop2 value exceeds maximum."
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "ExpMax"
         printf "Unable to check if value exceeds maximum. (Do2D)\r"
      endif

      FindValue/TEXT=(yokonum + "HardMin")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         variable hardmin2 = str2num(ExperimentalDetails[V_value][1])
         if(((srt2*divider)<hardmin2) || ((stop2*divider)<hardmin2))
            printf "Srt2 or stop2 value too low.  "
            printf "Yoko limited to %g V.  Aborting. (Do2D)\r", hardmin2
            Abort "Do2D: srt2 or stop2 value exceeds hardware minimum."
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokonum+ "HardMin"
         printf "Unable to check if value exceeds minimum. (Do2D)\r"
      endif
      FindValue/TEXT=(yokonum + "ExpMin")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         variable expmin2 = str2num(ExperimentalDetails[V_value][1])
         if((srt2<expmin2) || (stop2<expmin2))
            printf "Srt2 or stop2 value too low.  "
            printf "Limited to %g %s.  Aborting. (Do2D)\r", expmin2, ivarunits2
            Abort "Do2D: srt2 or stop2 value exceeds minimum."
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokonum+ "ExpMin"
         printf "Unable to check if value exceeds minimum. (Do2D)\r"
      endif

      FindValue/TEXT=(yokonum + "RateMax")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         maxrate2 = str2num(ExperimentalDetails[V_value][1])
         if(actualrate2>maxrate2)
            actualrate2 = maxrate2
            printf "Actualrate2 too fast: "
            printf "reducing actual rate to %g %s. (Do2D)\r", maxrate2, ivarunits2 + "/sec."
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", searchstr2 + "RateMax"
         printf "Sweep rate may be too fast. (Do2D)\r"
      endif
   endfor
elseif(StringMatch(searchstr2, "Idc"))
   FindValue/TEXT=("VtoI R")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      VtoIR = str2num(ExperimentalDetails[V_value][1])
   else
      Print "\"VtoI R\" not found in ExperimentalDetails.  Aborting. (Do2D)"
```

```
      Abort "Do2D: VtoI R not found."
   endif
   FindValue/TEXT=(searchstr2 + "Max")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      maxval2 = str2num(ExperimentalDetails[V_value][1])
      if(((srt2*1e-9*VtoIR)>maxval2) || ((srt2*1e-9*VtoIR)>maxval2))
         printf "Srt2 or stop2 value too high.  "
         printf "33220A limited to %g V.  Aborting. (Do2D)\r", maxval2
         Abort "Do2D: srt2 or stop2 value exceeds hardware maximum."
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr2 + "Max"
      printf "Unable to check if value exceeds maximum. (Do2D)\r"
   endif
   FindValue/TEXT=(searchstr2 + "Min")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      minval2 = str2num(ExperimentalDetails[V_value][1])
      if(((srt2*1e-9*VtoIR)<minval2) || ((srt2*1e-9*VtoIR)<minval2))
         printf "Srt2 or stop2 value too low.  "
         printf "33220A limited to %g V.  Aborting. (Do2D)\r", minval2
         Abort "Do2D: srt2 or stop2 value exceeds hardware minimum."
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr2+ "Min"
      printf "Unable to check if value exceeds minimum. (Do2D)\r"
   endif
   FindValue/TEXT=(searchstr2 + "RateMax")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      maxrate2 = str2num(ExperimentalDetails[V_value][1])
      if(rate2>maxrate2)
         rate2 = maxrate2
         printf "Rate2 too fast: reducing rate to %g %s. (Do2D)\r", maxrate2, ivarunits2 + "/sec."
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr2 + "RateMax"
      printf "Sweep rate may be too fast. (Do2D)\r"
   endif
elseif(StringMatch(searchstr2, "Vdc"))
   FindValue/TEXT=("VdcIDivider")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      VDivDC = str2num(ExperimentalDetails[V_value][1])
   else
      Print "\"VdcDivider\" not found in ExperimentalDetails.  Aborting. (Do2D)"
      Abort "Do2D: VdcDivider not found."
   endif
   FindValue/TEXT=(searchstr2 + "Max")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      maxval2 = str2num(ExperimentalDetails[V_value][1])
      if(((srt2*1e-3*VDivDC)>maxval2) || ((srt2*1e-3*VDivDC)>maxval2))
         printf "Srt2 or stop2 value too high.  "
         printf "33220A limited to %g V.  Aborting. (Do2D)\r", maxval2
         Abort "Do2D: srt2 or stop2 value exceeds hardware maximum."
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr2 + "Max"
      printf "Unable to check if value exceeds maximum. (Do2D)\r"
   endif
   FindValue/TEXT=(searchstr2 + "Min")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      minval2 = str2num(ExperimentalDetails[V_value][1])
      if(((srt2*1e-3*VDivDC)<minval2) || ((srt2*1e-3*VDivDC)<minval2))
         printf "Srt2 or stop2 value too low.  "
         printf "33220A limited to %g V.  Aborting. (Do2D)\r", minval2
```

```
         Abort "Do2D: srt2 or stop2 value exceeds hardware minimum."
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr2+ "Min"
      printf "Unable to check if value exceeds minimum. (Do2D)\r"
   endif
   FindValue/TEXT=(searchstr2 + "RateMax")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      maxrate2 = str2num(ExperimentalDetails[V_value][1])
      if(rate2>maxrate2)
         rate2 = maxrate2
         printf "Rate2 too fast: reducing rate to %g %s. (Do2D)\r", maxrate2, ivarunits2 + "/sec."
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr2 + "RateMax"
      printf "Sweep rate may be too fast. (Do2D)\r"
   endif
else
   FindValue/TEXT=(searchstr2 + "Max")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      maxval2 = str2num(ExperimentalDetails[V_value][1])
      if((srt2>maxval2) || (stop2>maxval2))
         printf "Srt2 or stop2 value too high.  "
         printf "Limited to %g %s.  Aborting. (Do2D)\r", maxval2, ivarunits2
         Abort "Do2D: srt2 or stop2 value exceeds maximum."
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr2 + "Max"
      printf "Unable to check if value exceeds maximum. (Do2D)\r"
   endif
   FindValue/TEXT=(searchstr2 + "Min")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      minval2 = str2num(ExperimentalDetails[V_value][1])
      if((srt2<minval2) || (stop2<minval2))
         printf "Srt2 or stop2 value too low.  "
         printf "Limited to %g %s.  Aborting. (Do2D)\r", minval2, ivarunits2
         Abort "Do2D: srt2 or stop2 value exceeds minimum."
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr2+ "Min"
      printf "Unable to check if value exceeds minimum. (Do2D)\r"
   endif
   FindValue/TEXT=(searchstr2 + "RateMax")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      maxrate2 = str2num(ExperimentalDetails[V_value][1])
      if(rate2>maxrate2)
         rate2 = maxrate2
         printf "Rate2 too fast: reducing rate to %g %s. (Do2D)\r", maxrate2, ivarunits2 + "/sec."
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr2 + "RateMax"
      printf "Sweep rate may be too fast. (Do2D)\r"
   endif
endif

//after this point the experiment changes, and will not revert after an abort

string recordstr = "", recordstr2 = "", recordstr3 = ""
sprintf recordstr, "%s: %s: Do2D(%s, ", date(), time(), idstr1
recordstr2 = "%g, %g, %g, %g, %s, ", srt1, stop1, ndiv1, rate1, idstr2
recordstr3 = "%g, %g, %g, %g, [%g, %g])\r", srt2, stop2, ndiv2, rate2, bkrate, delay
recordstr += recordstr2 + recordstr3
Notebook Record, selection={StartofFile, StartofFile}, text = recordstr
```

```
   SaveNotebook/O/P=savepath/S=6 Record as "Record.txt"

   FindValue/TEXT=("Independent")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      ExperimentalDetails[V_value][1] = idstr1 + ";" + idstr2
   else
      print "\Independent\" not found in ExperimentalDetails."
      print Unable to record independent variables. (Do2D)"
   endif

   string graphlist = WinList("Graph*", ";", "WIN:1") //list of open unnamed graphs
   variable numgraphs = ItemsInList(graphlist, ";"), k = 0
   for(k = 0; k<numgraphs; k+=1)
      KillWindow $StringFromList(k, graphlist, ";") //kill open unnamed graphs
   endfor
   string killlist = DataWaveList() //list of all data waves (with name matching proper format)
   variable killlength = ItemsInList(killlist, ";")
//kill all data waves not in use
   for(k=0; k<killlength; k+=1)
      //does not kill waves which are in graphs, tables, etc.
      KillWaves/Z $StringFromList(k, killlist, ";")
   endfor

   variable numpts1=ndiv1+1
   variable numpts2=ndiv2+1
   //update number of x and y points in Experimental Details
   FindValue/TEXT=("NumPoints X")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      ExperimentalDetails[V_value][1] = num2istr(numpts2)
   else
      Print "\"NumPoints X\" not found in ExperimentalDetails."
      Print "Unable to record number of x-dimension points. (Do2D)"
   endif
   FindValue/TEXT=("NumPoints Y")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      ExperimentalDetails[V_value][1] = num2istr(numpts1)
   else
      Print "\"NumPoints Y\" not found in ExperimentalDetails."
      Print "Unable to record number of y-dimension points. (Do2D)"
   endif

   variable nextwaveindex=nextwave(increment = 1)
   wave/T datatypedmm
   variable maxmeasurements = DimSize(datatypedmm, 0)
   make/O/T/N=(maxmeasurements) wavenamelist = ""
   string currentwavename = ""
   variable starttime = 0, datatime1 = 0, datatime2 = 0
   variable n = 0, m = 0, l = 0

   make/O/N=(maxmeasurements, 2) do2dSFs
   variable scale, offset

   string do2dDeVars = "", recordvalue = "", dmmlipa = "", currentdatatype = ""

   //update Start, Stop, and Rate in ExperimentalDetails
   if(StringMatch(searchstr1, "Vg"))
      for(y=0; y<checklength1; y+=1)
         yokonum = StringFromList(y, checklist1, ";")

         FindValue/TEXT=(yokonum+"Start")/TXOP=4 ExperimentalDetailsName
         if(V_Value > -1)
            sprintf recordvalue, "%g", srt1
            ExperimentalDetails[V_value][1] = recordvalue
```

```
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "Start"
         printf "Unable to record srt1 value. (Do2D)\r"
      endif
      FindValue/TEXT=(yokonum+"Stop")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         sprintf recordvalue, "%g", stop1
         ExperimentalDetails[V_value][1] = recordvalue
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "Stop"
         printf "Unable to record stop1 value. (Do2D)\r"
      endif
      FindValue/TEXT=(yokonum+"Rate")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         sprintf recordvalue, "%g", actualrate1
         ExperimentalDetails[V_value][1] = recordvalue
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "Rate"
         printf "Unable to record sweep rate1. (Do2D)\r"
      endif
   endfor
else
   FindValue/TEXT=(searchstr1+"Start")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      sprintf recordvalue, "%g", srt1
      ExperimentalDetails[V_value][1] = recordvalue
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr1 + "Start"
      printf "Unable to record srt1 value. (Do2D)\r"
   endif
   FindValue/TEXT=(searchstr1+"Stop")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      sprintf recordvalue, "%g", stop1
      ExperimentalDetails[V_value][1] = recordvalue
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr1 + "Stop"
      printf "Unable to record stop1 value. (Do2D)\r"
   endif
   FindValue/TEXT=(searchstr1+"Rate")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      sprintf recordvalue, "%g", rate1
      ExperimentalDetails[V_value][1] = recordvalue
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr1 + "Rate"
      printf "Unable to record sweep rate1. (Do2D)\r"
   endif
endif

if(StringMatch(searchstr2, "Vg"))
   for(y=0; y<checklength2; y+=1)
      yokonum = StringFromList(y, checklist2, ";")

      FindValue/TEXT=(yokonum+"Start")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         sprintf recordvalue, "%g", srt2
         ExperimentalDetails[V_value][1] = recordvalue
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "Start"
         printf "Unable to record srt2 value. (Do2D)\r"
      endif
      FindValue/TEXT=(yokonum+"Stop")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         sprintf recordvalue, "%g", stop2
```

```
         ExperimentalDetails[V_value][1] = recordvalue
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "Stop"
         printf "Unable to record stop2 value. (Do2D)\r"x
      endif
      FindValue/TEXT=(yokonum+"Rate")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         sprintf recordvalue, "%g", actualrate2
         ExperimentalDetails[V_value][1] = recordvalue
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "Rate"
         printf "Unable to record sweep rate2. (Do2D)\r"
      endif
   endfor
else
   FindValue/TEXT=(searchstr2+"Start")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      sprintf recordvalue, "%g", srt2
      ExperimentalDetails[V_value][1] = recordvalue
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr2 + "Start"
      printf "Unable to record srt2 value. (Do2D)\r"
   endif
   FindValue/TEXT=(searchstr2+"Stop")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      sprintf recordvalue, "%g", stop2
      ExperimentalDetails[V_value][1] = recordvalue
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr2 + "Stop"
      printf "Unable to record stop2 value. (Do2D)\r"
   endif
   FindValue/TEXT=(searchstr2+"Rate")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      sprintf recordvalue, "%g", rate2
      ExperimentalDetails[V_value][1] = recordvalue
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", searchstr2 + "Rate"
      printf "Unable to record sweep rate2. (Do2D)\r"
   endif
endif

string cscalelegend = ""
for(n=0; n < maxmeasurements; n+=1)
   currentdatatype = datatypedmm[n][0]
   if(!stringmatch(currentdatatype,"-")) //step through each dependent variable to be measured
      scale = 1 //for ReadDMM returning V
      offset = 0

      for(m=0; m<maxmeasurements; m+=1)
         if((m != n) && (stringmatch(currentdatatype, datatypedmm[m][0])))
            printf "Dependent variable %s chosen more than once ", currentdatatype
            printf "and will be overwritten.  Aborting. (Do2D)\r"
            Abort "Do2D: dependent variable chosen multiple times."
         endif
      endfor

      FindValue/TEXT=("DMMoffset" + datatypedmm[n][2])/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         offset += str2num(ExperimentalDetails[V_value][1])
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", "DMMoffset" + datatypedmm[n][2]
         printf "Offset not changed. (Do2D)\r"
      endif
```

```
      FindValue/TEXT=("DMMscale" + datatypedmm[n][2])/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         scale *= str2num(ExperimentalDetails[V_Value][1])
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", "DMMscale" + datatypedmm[n][2]
         printf "Scale not changed. (Do2D)\r"
      endif
      if(!stringmatch(datatypedmm[n][4], "0"))
         FindValue/TEXT=("LIsens" + datatypedmm[n][4])/TXOP=4 ExperimentalDetailsName
         if(V_Value > -1)
            scale *= str2num(ExperimentalDetails[V_value][1])/10
         else
            printf "\"%s\" not found in ExperimentalDetails.  ", "LIsens" + datatypedmm[n][3]
            printf "Scale not changed. (Do12D)\r"
         endif
      endif
      if(!stringmatch(datatypedmm[n][5], "0"))
         FindValue/TEXT=("PAamp" + datatypedmm[n][5])/TXOP=4 ExperimentalDetailsName
         if(V_Value > -1)
            scale /= str2num(ExperimentalDetails[V_value][1])
         else
            printf "\"%s\" not found in ExperimentalDetails.  ", "PAamp" + datatypedmm[n][4]
            printf "Scale not changed. (Do2D)\r"
         endif
      endif

      scale *= ScaleFactor(n)

      //record offset and scale values
      do2dSFs[n][0] = offset
      do2dSFs[n][1] = scale

      do2dDeVars += datatypedmm[n][0] + ";"
      //record DMM, lock-in, preamp chain
      dmmlipa += datatypedmm[n][2] + datatypedmm[n][4] + datatypedmm[n][5] + ";"

      sprintf currentwavename, "%s_%s_%s_%d", datatypedmm[n][0], idstr1, idstr2, nextwaveindex
      wavenamelist[n] = currentwavename //store wave name
      make/O/N=(numpts2, numpts1) $currentwavename = NaN //create wave
      SetScale/I y srt1, stop1, $currentwavename //set x scaling
      SetScale/I x srt2, stop2, $currentwavename //set y scaling
      Display; AppendImage $currentwavename //create graph
      ModifyImage $currentwavename ctab= {*,*,Terrain256,0} //set color scheme
      GraphStyle2D() //edit function to change graph style
      //create color scale
      ColorScale/C/N=cslegend/B=1/F=0/A=RC/X=0.00/Y=5.00/E=2 image = $currentwavename
      ColorScale/C/N=cslegend lblMargin=0
      cscalelegend = datatypedmm[n][0] + num2istr(nextwaveindex) + " (" + datatypedmm[n][1] +")"
      ColorScale/C/N=cslegend (cscalelegend)
      Label left, idstr1+ " (" + ivarunits1 + ")" //label y-axis
      Label bottom, idstr2+ " (" + ivarunits2 + ")" //label x-axis
      //move graph window
      movewindow/I (mod(n, 3)*3), (floor(n/3)*3), (mod(n, 3)*3 + 2.95), (floor(n/3)*3 + 2.2)
   endif
endfor
//update dependent variables in ExperimentalDetails
FindValue/TEXT=("Dependent")/TXOP=4 ExperimentalDetailsName
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = do2dDeVars
else
   Print "\"Dependent\" not found in ExperimentalDetails."
   Print "Unable to record dependent variables. (Do2D)"
endif
```

```
//update dmm/lock-in/preamp info in ExperimentalDetails
FindValue/TEXT=("DMMLIPA")/TXOP=4 ExperimentalDetailsName
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = dmmlipa
else
   Print "\"DMMLIPA\" not found in ExperimentalDetails."
   Print "Unable to record dependent variables. (Do2D)"
endif

FindValue/TEXT=("Date")/TXOP=4 ExperimentalDetailsName //update date
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = date()
else
   Print "\"Date\" not found in ExperimentalDetails.  Unable to record date. (Do2D)"
endif
FindValue/TEXT=("StartTime")/TXOP=4 ExperimentalDetailsName //update start times
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = time()
else
   Print "\"StartTime\" not found in ExperimentalDetails.  Unable to record time. (Do2D)"
endif

//save ExperimentalDetails to file
Duplicate/O/T/R=[][1] ExperimentalDetails do2dDetails
ExperimentalDetails[][DimSize(ExperimentalDetails, 1)-1] = do2dDetails[p]
Concatenate/T {ExperimentalDetailsName}, ExperimentalDetails
SaveDetails()

//constants for mixing chamber temperature fit
variable A0 = 411.7925271369486, A1 = -487.3966692536534, A2 = 233.1118681633504
variable A3 = -55.85454979733308, A4 = 6.690672513604219, A5 = -0.3205752792140923


//make waves for independent variables points
make/O/N=(numpts1) do2dInVals1
do2dInVals1=srt1+p*(stop1-srt1)/ndiv1
make/O/N=(numpts2) do2dInVals2
do2dInVals2=srt2+p*(stop2-srt2)/ndiv2

// data taking loop
make/O/N=(maxmeasurements) do2dcurrentvals = NaN
make/O do2dcurrentwave
SweepVal(idstr1, do2dInVals1[0], actualrate1, startB)
SweepVal(idstr2, do2dInVals2[0], actualrate2, startB)
//wait(1)
try
   starttime = stopMSTimer(-2)
   for(m=0; m<numpts2; m+=1) //slow axis loop
      SweepVal(idstr2, do2dInVals2[m], actualrate2, do2dInVals2[m-1]) //set slow axis variable
      if(!ParamIsDefault(bkrate)) //sweep back after every fast sweep
         if(m == 0)
            startB = do2dInVals1[0]
         else
            startB = do2dInVals1[Inf]
         endif
         SweepVal(idstr1, do2dInVals1[0], actualbkrate, startB)
         wait(delay)
         for(l=0; l<numpts1; l+=1)
            //set fast axis variable
            SweepVal(idstr1, do2dInVals1[l], actualrate1,do2dInVals1[l-1])
            TakeData(do2dcurrentvals) //take data
            for(n=0; n<maxmeasurements; n+=1) //step through each wave
               if(!stringmatch(datatypedmm[n][0], "-"))
```

```
            currentwavename = wavenamelist[n]
            wave do2dcurrentwave = $currentwavename
            //update wave with scaled data
            do2dcurrentwave[m][l] = (do2dcurrentvals[n]-do2dSFs[n][0])*do2dSFs[n][1]
            if((stringmatch(idstr1, "Time")) && (m==0)) //update fast axis scaling with time
               datatime1 = stopMSTimer(-2)
               SetScale/P y, 0, ((datatime1-starttime)*1e-6/(l+1)), $currentwavename
            endif
            if(stringmatch(datatypedmm[n][0], "Tmc")) //implement nonlinear temperature fit
               do2dcurrentwave[m][l] = A0 + A1 * log(do2dcurrentwave[m][l])
               do2dcurrentwave[m][l] += A2*(log(do2dcurrentwave[m][l]))^2
               do2dcurrentwave[m][l] += A3*(log(do2dcurrentwave[m][l]))^3
               do2dcurrentwave[m][l] += A4*(log(do2dcurrentwave[m][l]))^4
               do2dcurrentwave[m][l] += A5*(log(do2dcurrentwave[m][l]))^5
               do2dcurrentwave[m][l] = 10^do2dcurrentwave[m][l]
            endif
         endif
      endfor
   endfor
else //no back sweep (do zig-zag)
   wait(delay)
   if(mod(m,2)==0)
      for(l=0; l<numpts1; l+=1) //sweep start to stop
         SweepVal(idstr1, do2dInVals1[l], actualrate1, do2dInVals1[l-1])
         TakeData(do2dcurrentvals)
         for(n=0; n<maxmeasurements; n+=1)
            if(!stringmatch(datatypedmm[n][0], "-"))
               currentwavename = wavenamelist[n]
               wave do2dcurrentwave = $currentwavename
               do2dcurrentwave[m][l] = (do2dcurrentvals[n]-do2dSFs[n][0])*do2dSFs[n][1]
               if((stringmatch(idstr1, "time")) && (m==0))
                  datatime1 = stopMSTimer(-2)
                  SetScale/P y, 0, ((datatime1-starttime)*1e-6/(l+1)), $currentwavename
               endif
               //implement nonlinear temperature fit
               if(stringmatch(datatypedmm[n][0], "Tmc"))
                  do2dcurrentwave[m][l] = A0 + A1 * log(do2dcurrentwave[m][l])
                  do2dcurrentwave[m][l] += A2*(log(do2dcurrentwave[m][l]))^2
                  do2dcurrentwave[m][l] += A3*(log(do2dcurrentwave[m][l]))^3
                  do2dcurrentwave[m][l] += A4*(log(do2dcurrentwave[m][l]))^4
                  do2dcurrentwave[m][l] += A5*(log(do2dcurrentwave[m][l]))^5
                  do2dcurrentwave[m][l] = 10^do2dcurrentwave[m][l]
               endif
            endif
         endfor
      endfor
   else
      for(l=numpts1-1; l>-1; l-=1) //sweep stop to start
         SweepVal(idstr1, do2dInVals1[l], actualrate1, do2dInVals1[l+1])
         TakeData(do2dcurrentvals)
         for(n=0; n<maxmeasurements; n+=1)
            if(!stringmatch(datatypedmm[n][0], "-"))
               currentwavename = wavenamelist[n]
               wave do2dcurrentwave = $currentwavename
               do2dcurrentwave[m][l] = (do2dcurrentvals[n]-do2dSFs[n][0])*do2dSFs[n][1]
               if((stringmatch(idstr1, "Time")) && (m==0))
                  datatime1 = stopMSTimer(-2)
                  SetScale/P y, 0, ((datatime1-starttime)*1e-6/(l+1)), $currentwavename
               endif
               //implement nonlinear temperature fit
               if(stringmatch(datatypedmm[n][0], "Tmc"))
                  do2dcurrentwave[m][l] = A0 + A1 * log(do2dcurrentwave[m][l])
                  do2dcurrentwave[m][l] += A2*(log(do2dcurrentwave[m][l]))^2
```

```
                        do2dcurrentwave[m][1] += A3*(log(do2dcurrentwave[m][1]))^3
                        do2dcurrentwave[m][1] += A4*(log(do2dcurrentwave[m][1]))^4
                        do2dcurrentwave[m][1] += A5*(log(do2dcurrentwave[m][1]))^5
                        do2dcurrentwave[m][1] = 10^do2dcurrentwave[m][1]
                     endif
                  endif
               endfor
            endfor
         endif
      endif
      if(StringMatch(idstr2, "Time")) //update slow axis scaling with time
         datatime2 = stopMSTimer(-2)
         for(n=0; n<maxmeasurements; n+=1)
            if(!stringmatch(datatypedmm[n][0], "-"))
               currentwavename = wavenamelist[n]
               SetScale/P x, 0, ((datatime2-starttime)*1e-6/(m+1)), $currentwavename
            endif
         endfor
      endif
      DoUpdate //Comment this line if no data will be displayed during acquisition JBM060828
   endfor
   datatime2 = stopMSTimer(-2)
catch
   DoUpdate
endtry
AllDMMLocal() //do we want this?
try //save waves and update ExperimentalDetails even if function aborted
   for(n=0; n<maxmeasurements; n+=1) //save each wave as an Igor binary file
      if(!stringmatch(datatypedmm[n][0], "-"))
         currentwavename = wavenamelist[n]
         Save/P=binarypath $currentwavename
         Save/P=backuppath $currentwavename
      endif
   endfor

   //update Start, Stop, and Rate in ExperimentalDetails
   if(StringMatch(searchstr1, "Vg"))
      for(y=0; y<checklength1; y+=1)
         yokonum = StringFromList(y, checklist1, ";")

         FindValue/TEXT=(yokonum+"Start")/TXOP=4 ExperimentalDetailsName
         if(V_Value > -1)
            sprintf recordvalue, "%g", do2dInVals1[1]
            ExperimentalDetails[V_value][1] = recordvalue
         else
            printf "Warning: unable to update %s after sweep. (Do2D)\r", yokonum + "Start"
         endif
         FindValue/TEXT=(yokonum+"Stop")/TXOP=4 ExperimentalDetailsName
         if(V_Value > -1)
            sprintf recordvalue, "%g", do2dInVals1[1]
            ExperimentalDetails[V_value][1] = recordvalue
         else
            printf "Warning: unable to update %s after sweep. (Do2D)\r", yokonum + "Stop"
         endif
         FindValue/TEXT=(yokonum+"Rate")/TXOP=4 ExperimentalDetailsName
         if(V_Value > -1)
            ExperimentalDetails[V_value][1] = "0"
         else
            printf "Warning: unable to update %s after sweep. (Do2D)\r", yokonum + "Rate"
         endif
      endfor
   else
      FindValue/TEXT=(searchstr1+"Start")/TXOP=4 ExperimentalDetailsName
```

```
   if(V_Value > -1)
      sprintf recordvalue, "%g", do2dInVals1[l]
      ExperimentalDetails[V_value][1] = recordvalue
   else
      printf "Warning: unable to update %s after sweep. (Do2D)\r", searchstr1 + "Start"
   endif
   FindValue/TEXT=(searchstr1+"Stop")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      sprintf recordvalue, "%g", do2dInVals1[l]
      ExperimentalDetails[V_value][1] = recordvalue
   else
      printf "Warning: unable to update %s after sweep. (Do2D)\r", searchstr1 + "Stop"
   endif
   FindValue/TEXT=(searchstr1+"Rate")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      ExperimentalDetails[V_value][1] = "0"
   else
      printf "Warning: unable to update %s after sweep. (Do2D)\r", searchstr1 + "Rate"
   endif
endif

if(StringMatch(searchstr2, "Vg"))
   for(y=0; y<checklength2; y+=1)
      yokonum = StringFromList(y, checklist2, ";")

      FindValue/TEXT=(yokonum+"Start")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         sprintf recordvalue, "%g", do2dInVals2[m]
         ExperimentalDetails[V_value][1] = recordvalue
      else
         printf "Warning: unable to update %s after sweep. (Do2D)\r", yokonum + "Start"
      endif
      FindValue/TEXT=(yokonum+"Stop")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         sprintf recordvalue, "%g", do2dInVals2[m]
         ExperimentalDetails[V_value][1] = recordvalue
      else
         printf "Warning: unable to update %s after sweep. (Do2D)\r", yokonum + "Stop"
      endif
      FindValue/TEXT=(yokonum+"Rate")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         ExperimentalDetails[V_value][1] = "0"
      else
         printf "Warning: unable to update %s after sweep. (Do2D)\r", yokonum + "Rate"
      endif
   endfor
else
   FindValue/TEXT=(searchstr2+"Start")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      sprintf recordvalue, "%g", do2dInVals2[m]
      ExperimentalDetails[V_value][1] = recordvalue
   else
      printf "Warning: unable to update %s after sweep. (Do2D)\r", searchstr2 + "Start"
   endif
   FindValue/TEXT=(searchstr2+"Stop")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      sprintf recordvalue, "%g", do2dInVals2[m]
      ExperimentalDetails[V_value][1] = recordvalue
   else
      printf "Warning: unable to update %s after sweep. (Do2D)\r", searchstr2 + "Stop"
   endif
   FindValue/TEXT=(searchstr2+"Rate")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
```

```
            ExperimentalDetails[V_value][1] = "0"
         else
            printf "Warning: unable to update %s after sweep. (Do2D)\r", searchstr2 + "Rate"
         endif
      endif
   catch
      Print "Warning: function aborted, data may not be saved."
      Print "Check that all files are properly saved. (Do2D)"
   endtry
   printf "Created waves #%d, finished at %s, ",nextwaveindex,time()
   printf "elapsed time %6.3f min.\r",((datatime2-starttime)*1e-6/60)
   sprintf recordstr, "Created waves #%d, finished at %s, ",nextwaveindex,time()
   sprintf recordstr2, "elapsed time %6.3f min.\r",((datatime2-starttime)*1e-6/60)
   recordstr += recordstr2
   Notebook Record, selection={StartofFile, StartofFile}, text = recordstr
   SaveNotebook/O/P=savepath/S=6 Record as "Record.txt"
end

//sweeps gates until Vg reaches stop or measured resistance exceeds maxR
function GateTest(gatestr, stop, numdivs, rate, datatypenum, maxR)
   string gatestr       // gate numbers in string
   variable stop        // maximum magnitude of gate voltage, also used to calculate point spacing
   variable numdivs     // number of points minus 1 (if run until stop)
   variable rate        // sweep rate per second
   variable datatypenum // which datatype to check resistance
   variable maxR     // maximum resistance (in h/e^2) allowed before sweep stops

   rate = abs(rate) //make sure rate is positive

   wave/T ExperimentalDetails
   Duplicate/O/R=[][0] ExperimentalDetails, ExperimentalDetailsName

   wave/T datatypedmm
   variable maxmeasurements = DimSize(datatypedmm, 0)
   if((datatypenum > maxmeasurements) || (datatypenum < 1))
      printf "Data type %g is out of range; ", datatypenum, maxmeasurements -1
      printf "must be an integer [1, %d].  Aborting. (GateTest)\r"
      Abort "GateTest: datatypenum out of range."
   endif
   if(!stringmatch(datatypedmm[datatypenum-1][0], "R*"))
      printf "Dependent variable selected for data type #%d is not a resistance.  ", datatypenum
      printf "Aborting. (GateTest)\r"
      Abort "GateTest: datatypenum does not correspond to a resistance."
   endif

   if(GrepString(gatestr,"^\d+$")==0)
      printf "String %s contains nonnumeral characters; ", gatestr
      printf "only use concatenated gate numbers.  Aborting.  (GateTest)\r"
      Abort "GateTest: gatestr contains nonnumeral characters."
   endif

   string idstr = "Vg" + gatestr
   variable numgates = strlen(gatestr)
   variable y = 0
   string yokonum = "", checklist = ""
   for(y=0; y<numgates; y+=1)
      checklist += "Vg" + gatestr[y] + ";"
   endfor

   string searchstr = "Vg"
   wave/T InVarWave
   FindValue/TEXT=searchstr/TXOP=4 InVarWave //check that Vg is a valid independent variable
   if((V_value == -1) || (V_value >= DimSize(InVarWave, 0)))
```

```
   printf "%s is not a valid independent variable ", searchstr
   printf "(or is not included in InVarWave).  Aborting. (GateTest)\r"
   Abort "GateTest: invalid independent variable."
else
   string ivarunits = InVarWave[V_value][1]
endif

variable maxval = 0, minval = 0, maxrate = 0
variable start = 0
//checks inputs against Max, Min, and RateMax
variable divider = Inf
variable actualrate = rate * numgates //account for sweeping multiple gates
for(y=0; y<numgates; y+=1) //check each specified gate
   yokonum = StringFromList(y, checklist, ";")

   //make sure rate does not exceed RateMax
   FindValue/TEXT=(yokonum + "RateMax")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      maxrate = str2num(ExperimentalDetails[V_value][1])
      if(actualrate>maxrate)
         actualrate = maxrate
         printf "Actualrate too fast: "
         printf "reducing actual rate to %g %s. (GateTest)\r", maxrate, ivarunits + "/sec"
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "RateMax"
      printf "Sweep rate may be too fast. (GateTest)\r"
   endif

   FindValue/TEXT=(yokonum + "Divider")/TXOP=4 ExperimentalDetailsName //get divider value
   if(V_Value > -1)
      if(str2num(ExperimentalDetails[V_value][1]) > 0)
         divider = str2num(ExperimentalDetails[V_value][1])
      else
         printf "\"%s\" value not positive.  Aborting. (GateTest)\r", yokonum + "Divider"
         Abort "GateTest: Vg#Divider not positive."
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "Divider"
      printf "Aborting. (GateTest)\r"
      Abort "GateTest: cannot find Vg#Divider in ExperimentalDetails."
   endif

   //check hardware (Yoko) maximum
   FindValue/TEXT=(yokonum + "HardMax")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      variable hardmax = str2num(ExperimentalDetails[V_value][1])
      if(((start*divider)>hardmax) || ((stop*divider)>hardmax))
         printf "Start or stop value too high.  "
         printf "Yoko limited to %g V.  Aborting. (GateTest)\r", hardmax
         Abort "GateTest: start or stop value exceeds hardware maximum."
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "HardMax"
      printf "Unable to check if value exceeds maximum. (GateTest)\r"
   endif
   v
   FindValue/TEXT=(yokonum + "ExpMax")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      variable expmax = str2num(ExperimentalDetails[V_value][1])
      if((start>expmax) || (stop>expmax))
         printf "Start or stop value too high.  "
         printf "Limited to %g %s.  Aborting. (GateTest)\r", expmax, ivarunits
```

```
         Abort "GateTest: start or stop value exceeds maximum."
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "ExpMax"
      printf "Unable to check if value exceeds maximum. (GateTest)\r"
   endif

   //check hardware (Yoko) minimum
   FindValue/TEXT=(yokonum + "HardMin")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      variable hardmin = str2num(ExperimentalDetails[V_value][1])
      if(((start*divider)<hardmin) || ((stop*divider)<hardmin))
         printf "Start or stop value too low.  ", hardmin
         printf "Yoko limited to %g V.  Aborting. (GateTest)\r"
         Abort "GateTest: start or stop value exceeds hardware minimum."
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "HardMin"
      printf "Unable to check if value exceeds minimum. (GateTest)\r"
   endif
   //check experimental (device) minimum
   FindValue/TEXT=(yokonum + "ExpMin")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      variable expmin = str2num(ExperimentalDetails[V_value][1])
      if((start<expmin) || (stop<expmin))
         printf "Start or stop value too low.  "
         printf "Limited to %g %s.  Aborting. (GateTest)\r", expmin, ivarunits
         Abort "GateTest: start or stop value exceeds  minimum."
      endif
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "ExpMin"
      printf "Unable to check if value exceeds minimum. (GateTest)\r"
   endif
endfor

//after this point the experiment changes, and will not revert after an abort

string recordstr = "", recordstr2 = ""
sprintf recordstr, "%s: %s: GateTest(%s,", date(), time(), gatestr,
sprintf recordstr2, "%g, %g, %g, %g, %g)\r" stop, numdivs, rate, datatypenum, maxR
recordstr += recordstr2
Notebook Record, selection={StartofFile, StartofFile}, text = recordstr
SaveNotebook/O/P=savepath/S=6 Record as "Record.txt"

FindValue/TEXT=("Independent")/TXOP=4 ExperimentalDetailsName
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = idstr
else
   print "\"Independent\" not found in ExperimentalDetails."
   print Unable to record independent variable. (GateTest)"
endif

NVAR KeepGraphsFlag
string graphlist = WinList("Graph*", ";", "WIN:1") //get list of unnamed open graphs
variable numgraphs = ItemsInList(graphlist, ";"), k = 0
if(KeepGraphsFlag == 0)
   for(k = 0; k<numgraphs; k+=1)
      KillWindow $StringFromList(k, graphlist, ";") //kill all unnamed open graphs
   endfor
   string killlist = DataWaveList() //list of all data waves (with name matching proper format)
   variable killlength = ItemsInList(killlist, ";")
   //kill all data waves not in use
   for(k=0; k<killlength; k+=1)
```

```
      //does not kill waves which are in graphs, tables, etc.
      KillWaves/Z $StringFromList(k, killlist, ";")
   endfor
elseif(KeepGraphsFlag == 1)
   string wlist = "" //list of waves in graph
   string commandstr = ""
   variable numwaves = 0 //number of waves in graph
   variable waveappended = 0 //indicates if new wave could be appended to existing graph
else
   printf "KeepGraphsFlag value of %g invalid.  ", KeepGraphsFlag
   printf "Must be 0 or 1.  Aborting. (GateTest)\r"
   Abort "GateTest: invalid value of KeepGraphsFlag"
endif

variable numpts=numdivs+1
//update number of x and y points in ExperimentalDetails
FindValue/TEXT=("NumPoints X")/TXOP=4 ExperimentalDetailsName
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = num2istr(numpts)
else
   Print "\"NumPoints X\" not found in ExperimentalDetails."
   Print Unable to record number of x-dimension points. (GateTest)"
endif
FindValue/TEXT=("NumPoints Y")/TXOP=4 ExperimentalDetailsName
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = "0"
else
   Print "\"NumPoints Y\" not found in ExperimentalDetails."
   Print "Unable to record number of y-dimension points. (GateTest)"
endif

variable nextwaveindex=nextwave(increment = 1) //increment data index by 1
make/O/T/N=(maxmeasurements) wavenamelist = ""
string currentwavename = ""
variable starttime = 0, datatime = 0
variable n = 0, m = 0

//wave to keep track of offsets and scale factors
make/O/N=(maxmeasurements, 2) gatetestSFs
variable scale, offset

string gatetestDeVars = "", recordvalue = "", dmmlipa = "", currentdatatype = ""
//update Start, Stop, and Rate values in ExperimentalDetails
for(y=0; y<numgates; y+=1)
   yokonum = StringFromList(y, checklist, ";")
   FindValue/TEXT=(yokonum+"Start")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      sprintf recordvalue, "%g", start
      ExperimentalDetails[V_value][1] = recordvalue
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "Start"
      printf "Unable to record start value. (GateTest)\r"
   endif
   FindValue/TEXT=(yokonum+"Stop")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      sprintf recordvalue, "%g", stop
      ExperimentalDetails[V_value][1] = recordvalue
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "Stop"
      printf "Unable to record stop value. (GateTest)\r"
   endif
   FindValue/TEXT=(yokonum+"Rate")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
```

```
      sprintf recordvalue, "%g", actualrate
      ExperimentalDetails[V_value][1] = recordvalue
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", yokonum + "Rate"
      printf "Unable to record sweep rate. (GateTest)\r"
   endif
endfor

variable windowleft, windowtop, windowright, windowbottom
for(n=0; n < maxmeasurements; n+=1) //step though each dependent variable to be measured
   currentdatatype = datatypedmm[n][0]
   if(!stringmatch(currentdatatype,"-"))
      scale = 1 //for ReadDMM returning V
      offset = 0

      for(m=0; m<maxmeasurements; m+=1)
         if((m != n) && (stringmatch(currentdatatype, datatypedmm[m][0])))
            printf "Dependent variable %s chosen more than once ", currentdatatype
            printf "and will be overwritten.  Aborting. (GateTest)\r"
            Abort "GateTest: dependent variable chosen multiple times."
         endif
      endfor

      //get dmm offset (y-intercept)
      FindValue/TEXT=("DMMoffset" + datatypedmm[n][2])/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         offset += str2num(ExperimentalDetails[V_value][1])
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", "DMMoffset" + datatypedmm[n][2]
         printf "Offset not changed. (GateTest)\r"
      endif
      //get dmm scale factor
      FindValue/TEXT=("DMMscale" + datatypedmm[n][2])/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         scale *= str2num(ExperimentalDetails[V_Value][1])
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", "DMMscale" + datatypedmm[n][2]
         printf "Scale not changed. (GateTest)\r"
      endif
      if(!stringmatch(datatypedmm[n][4], "0"))
         //get lock-in sensitivity
         FindValue/TEXT=("LIsens" + datatypedmm[n][4])/TXOP=4 ExperimentalDetailsName
         if(V_Value > -1)
            scale *= str2num(ExperimentalDetails[V_value][1])/10
         else
            printf "\"%s\" not found in ExperimentalDetails.  ", "LIsens" + datatypedmm[n][3]
            printf "Scale not changed. (GateTest)\r"
         endif
      endif
      if(!stringmatch(datatypedmm[n][5], "0"))
         //get pre-amp amplification
         FindValue/TEXT=("PAamp" + datatypedmm[n][5])/TXOP=4 ExperimentalDetailsName
         if(V_Value > -1)
            scale /= str2num(ExperimentalDetails[V_value][1])
         else
            printf "\"%s\" not found in ExperimentalDetails.  ", "PAamp" + datatypedmm[n][4]
            printf "Scale not changed. (GateTest)\r"
         endif
      endif

      scale *= ScaleFactor(n)

      //update gatetestSFs wave
```

```
      gatetestSFs[n][0] = offset
      gatetestSFs[n][1] = scale

      gatetestDeVars += currentdatatype+ ";" //keep track of dependent variables to be measured
      //record DMM, lock-in, preamp chain
      dmmlipa += datatypedmm[n][2] + datatypedmm[n][4] + datatypedmm[n][5] + ";"

      sprintf currentwavename, "%s_%s_%d", datatypedmm[n][0], idstr, nextwaveindex
      wavenamelist[n] = currentwavename //keep list of wave names
      make/O/N=(numpts) $currentwavename = NaN //make the wave
      SetScale/I x start, stop, $currentwavename //set x scaling
      if(KeepGraphsFlag == 0) //make new graphs for next set of data
         Display $currentwavename //create graph
         Label bottom, idstr + " (" + ivarunits + ")" //label x-axis
         Label left, datatypedmm[n][0] + " (" + datatypedmm[n][1] +")" //label y-axis
         GraphStyle() //edit function to change graph style
         //move graph window
         movewindow/I (mod(n, 3)*3), (floor(n/3)*3), (mod(n, 3)*3 + 2.95), (floor(n/3)*3 + 2.2)
      elseif(KeepGraphsFlag == 1) //display data on old graphs
         waveappended = 0
         for(m=0; m<numgraphs; m+=1) //check each graph for match of devar
            //writes list of waves in graph to W_WaveList
            commandstr = "GetWindow " + StringFromList(m, graphlist, ";") + ", wavelist"
            Execute/Q commandstr
            wlist = TWave2Str(W_WaveList) //generate wlist
            numwaves = ItemsInList(wlist, ";") //number of waves on graph
            for(k=0; k<numwaves; k+=1) //check each wave to see if match devar
               //if match, append new trace
               if(stringmatch(StringFromList(k, wlist, ";"), currentdatatype + "_*"))
                  commandstr = "AppendToGraph/W=" + StringFromList(m, graphlist, ";")
                  commandstr += " " + currentwavename
                  Execute/Q commandstr
                  waveappended = 1
                  break
               endif
            endfor
         endfor
         if(waveappended == 0)
            Display $currentwavename //create graph
            Label bottom, idstr + " (" + ivarunits + ")" //label x-axis
            Label left, datatypedmm[n][0] + " (" + datatypedmm[n][1] +")" //label y-axis
            GraphStyle() //edit function to change graph style
            windowleft = (mod(n+numgraphs, 3)*3)
            windowtop = (floor((n+numgraphs)/3)*3)
            windowright = (mod(n+numgraphs, 3)*3 + 2.95)
            windowbottom = (floor((n+numgraphs)/3)*3 + 2.2)
            movewindow/I windowleft, windowtop, windowright, windowbottom //move graph window
         endif
      endif
   endif
endfor
//update dependent variable in ExperimentalDetails
FindValue/TEXT=("Dependent")/TXOP=4 ExperimentalDetailsName
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = gatetestDeVars
else
   Print "\"Dependent\" not found in ExperimentalDetails."
   Print "Unable to record dependent variables. (GateTest)"
endif
//update dmm/lock-in/preamp info in ExperimentalDetails
FindValue/TEXT=("DMMLIPA")/TXOP=4 ExperimentalDetailsName
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = dmmlipa
```

```
else
   Print "\"DMMLIPA\" not found in ExperimentalDetails."
   Print "Unable to record dependent variables. (GateTest)"
endif

FindValue/TEXT=("Date")/TXOP=4 ExperimentalDetailsName //update date
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = date()
else
   Print "\"Date\" not found in ExperimentalDetails.  Unable to record date. (GateTest)"
endif
FindValue/TEXT=("StartTime")/TXOP=4 ExperimentalDetailsName //update start time
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = time()
else
   Print "\"StartTime\" not found in ExperimentalDetails.  Unable to record time. (GateTest)"
endif

//save ExperimentalDetails to file
Duplicate/O/T/R=[][1] ExperimentalDetails gatetestDetails
ExperimentalDetails[][DimSize(ExperimentalDetails, 1)-1] = gatetestDetails[p]
Concatenate/T {ExperimentalDetailsName}, ExperimentalDetails
SaveDetails()

//constants for mixing chamber temperature fit
variable A0 = 411.7925271369486, A1 = -487.3966692536534, A2 = 233.1118681633504
variable A3 = -55.85454979733308, A4 = 6.690672513604219, A5 = -0.3205752792140923

make/O/N=(numpts) gatetestInVals //wave of independent variable points
gatetestInVals=start+p*(stop-start)/numdivs

// data taking loop
make/O/N=(maxmeasurements) gatetestcurrentvals = NaN
make/O gatetestcurrentwave
SweepVal(idstr, gatetestInVals[0], actualrate, 0)
//wait(1)
variable checkval = 0
try
   starttime = stopMSTimer(-2)
   for(m=0; m<numpts; m+=1)
      //set independent variable
      SweepVal(idstr, gatetestInVals[m], actualrate, gatetestInVals[m-1])
      TakeData(gatetestcurrentvals) //record data from dmms
      for(n=0; n<maxmeasurements; n+=1) //step through waves being taken
         if(!stringmatch(datatypedmm[n][0], "-"))
            currentwavename = wavenamelist[n]
            wave gatetestcurrentwave = $currentwavename
            //update wave with with scaled data
            gatetestcurrentwave[m] = (gatetestcurrentvals[n]-gatetestSFs[n][0])*gatetestSFs[n][1]
            if(stringmatch(idstr, "Time")) //scale x-axis with time
               datatime = stopMSTimer(-2)
               SetScale/P x, 0, ((datatime-starttime)*1e-6/(m+1)), $currentwavename
            endif
            if(stringmatch(datatypedmm[n][0], "Tmc")) //implement nonlinear temperature fit
               gatetestcurrentwave[m] = A0 + A1 * log(gatetestcurrentwave[m])
               gatetestcurrentwave[m] += A2*(log(gatetestcurrentwave[m]))^2
               gatetestcurrentwave[m] += A3*(log(gatetestcurrentwave[m]))^3
               gatetestcurrentwave[m] += A4*(log(gatetestcurrentwave[m]))^4
               gatetestcurrentwave[m] += A5*(log(gatetestcurrentwave[m]))^5
               gatetestcurrentwave[m] = 10^gatetestcurrentwave[m]
            endif
         endif
      endfor
```

```
        DoUpdate //Comment this line if no data will be displayed during acquisition JBM060828

        checkval = (gatetestcurrentvals[datatypenum-1]-gatetestSFs[datatypenum-1][0])
        if(abs(checkval*gatetestSFs[datatypenum-1][1]) >= maxR)
           break
        endif
     endfor
     datatime = stopMSTimer(-2)
  catch
     datatime = stopMSTimer(-2)
     DoUpdate
  endtry
  AllDMMLocal() //do we want this?
  try //save waves and update ExperimentalDetails even if function aborted
     for(n=0; n<maxmeasurements; n+=1)
        if(!stringmatch(datatypedmm[n][0], "-")) //save waves as Igor binary files
           currentwavename = wavenamelist[n]
           Save/P=binarypath $currentwavename
           Save/P=backuppath $currentwavename
        endif
     endfor

     //update Start, Stop, and Rate to current values (may not match input if function was aborted)
     for(y=0; y<numgates; y+=1)
        yokonum = StringFromList(y, checklist, ";")
        FindValue/TEXT=(yokonum+"Start")/TXOP=4 ExperimentalDetailsName
        if(V_Value > -1)
           sprintf recordvalue, "%g", gatetestInVals[m]
           ExperimentalDetails[V_value][1] = recordvalue
        else
           printf "Warning: unable to update %s after sweep. (GateTest)\r", yokonum + "Start"
        endif
        FindValue/TEXT=(yokonum+"Stop")/TXOP=4 ExperimentalDetailsName
        if(V_Value > -1)
           sprintf recordvalue, "%g", gatetestInVals[m]
           ExperimentalDetails[V_value][1] = recordvalue
        else
           printf "Warning: unable to update %s after sweep. (GateTest)\r", yokonum + "Stop"
        endif
        FindValue/TEXT=(yokonum+"Rate")/TXOP=4 ExperimentalDetailsName
        if(V_Value > -1)
           ExperimentalDetails[V_value][1] = "0"
        else
           printf "Warning: unable to update %s after sweep. (GateTest)\r", yokonum + "Rate"
        endif
     endfor
  catch
     Print "Warning: function aborted, data may not be saved."
     Print "Check that all files are properly saved. (GateTest)"
  endtry
  printf "Created waves #%d, finished at %s, ",nextwaveindex,time(),
  printf "elapsed time %6.3f min.\r"((datatime-starttime)*1e-6/60)
  sprintf recordstr, "Created waves #%d, finished at %s, " ,nextwaveindex,time()
  sprintf recordstr2, "elapsed time %6.3f min.\r",((datatime-starttime)*1e-6/60)
  recordstr += recordstr2
  Notebook Record, selection={StartofFile, StartofFile}, text = recordstr
  SaveNotebook/O/P=savepath/S=6 Record as "Record.txt"
end

//sweeps a specified combination of gate voltages
function Vg1D(numdivs)
  variable numdivs     // number of points minus 1
```

```
wave/T ExperimentalDetails
Duplicate/O/R=[][0] ExperimentalDetails, ExperimentalDetailsName

wave/T InVarWave
string idstr = "Vg"
FindValue/TEXT="Vg"/TXOP=4 InVarWave //check if idstr is a valid independent variable
if((V_value == -1) || (V_value >= DimSize(InVarWave, 0)))
   Print "Vg is not a valid independent variable (or is not included in InVarWave)."
   Print "Aborting. (Vg1D)"
   Abort "Vg1D: Vg not in InVarWave."
else
   string ivarunits = InVarWave[V_value][1]
endif

string Vglist = ""
variable num = 0
for(num=0; num<=9; num+=1)
   //get divider value
   FindValue/TEXT=("Vg" + num2istr(num) + "Divider")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         if(str2num(ExperimentalDetails[V_value][1]) > 0)
            Vglist += "Vg" + num2istr(num) + ";"
         endif
      endif
endfor
variable numgates = ItemsInList(Vglist, ";")
if(numgates <= 0)
   Print "No valid \"Vg#Divider\" found in ExperimentalDetails."
   Print "Must have at least one gate to sweep.  Aborting. (Vg1D)"
   Abort "Vg1D: no gates found."
endif

make/o/n=9 Vg1DUse
Vg1DUse = 0
NewPanel/N=Vg1DPanel/W=(400, 400, 750, 425 + numgates * 25) as "Vg1DInfo"
string commandstr = "", commandstr1 = "", commandstr2 = "", commandstr3 = ""
string commandstr4 = "", commandstr5 = "", commandstr6 = "", commandstr7 = ""
string commandstrb = "", commandstr1b = "", commandstr3b = ""
string commandstr5b = "", commandstr7b = ""
string Vgname = ""
for(num=0; num<numgates; num+=1)
   Vgname = StringFromList(num, Vglist, ";")
   sprintf commandstr "variable/G V_start%s = NaN, ", Vgname
   sprintf commandstrb "V_stop%s = NaN, V_rate%s = NaN", Vgname, Vgname
   commandstr += commandstrb
   sprintf commandstr1 "CheckBox use%s, pos={0,%d}, ", Vgname, num*25
   sprintf commandstr1b "size={14,14}, proc=Vg1DCheck, title=\"%s\"", Vgname
   commandstr1 += commandstr1b
   sprintf commandstr2 "SetVariable start%s, pos={100,%d}, ", Vgname, num*25
   commandstr2 += "size={41,31}, noproc, title=\"Start\""
   sprintf commandstr3 "SetVariable start%s, bodyWidth=60, ", Vgname
   sprintf commandstr3b "disable=1, format=%s, value=V_start%s", "\"%g\"", Vgname
   commandstr3 += commandstr3b
   sprintf commandstr4 "SetVariable stop%s, pos={200,%d}, ", Vgname, num*25
   commandstr4 += "size={41,31}, noproc, title=\"Stop\""
   sprintf commandstr5 "SetVariable stop%s, bodyWidth=60, ", Vgname
   sprintf commandstr5b "disable=1, format=%s, value=V_stop%s", "\"%g\"", Vgname
   commandstr5 += commandstr5b
   sprintf commandstr6 "SetVariable rate%s, pos={300,%d}, ", Vgname, num*25
   commandstr6 += "size={41,31}, proc=Vg1DRate, title=\"Rate\""
   sprintf commandstr7 "SetVariable rate%s, bodyWidth=60, ", Vgname
   sprintf commandstr7b "disable=1, format=%s, value=V_rate%s", "\"%g\"", Vgname
   commandstr7 += commandstr7b
```

```
   Execute/Q commandstr
   Execute/Q commandstr1
   Execute/Q commandstr2
   Execute/Q commandstr3
   Execute/Q commandstr4
   Execute/Q commandstr5
   Execute/Q commandstr6
   Execute/Q commandstr7
endfor

Button doCancel, pos={50,num*25}, size={60,17}, proc=Vg1DCancel, fstyle=1, title="Cancel"
Button doValidate, pos={150,num*25}, size={60,17}, proc=Vg1DValidate, fstyle=1, title="Validate"
Button doRun, pos={250,num*25}, size={60,17}, proc=Vg1DRun, fstyle=1, disable=2, title="Run"

PauseForUser Vg1DPanel

//after this point the experiment changes, and will not revert after an abort

string recordstr = ""
sprintf recordstr, "%s: %s: Vg1D(%g)\r", date(), time(), numdivs
Notebook Record, selection={StartofFile, StartofFile}, text = recordstr
SaveNotebook/O/P=savepath/S=6 Record as "Record.txt"

wave Vg1DUse
string Vgstr = "", Vgnumlist = ""
for(num=0; num<numgates; num+=1)
   Vgstr = StringFromList(num, Vglist, ";")
   if(Vg1DUse[str2num(Vgstr[2])] == 1)
      Vgnumlist += Vgstr[2]
   endif
endfor
variable numVgs = strlen(Vgnumlist)

idstr += Vgnumlist + "c"
FindValue/TEXT=("Independent")/TXOP=4 ExperimentalDetailsName
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = idstr
else
   print "\"Independent\" not found in ExperimentalDetails."
   print "Unable to record independent variable. (Vg1D)"
endif

NVAR KeepGraphsFlag
string graphlist = WinList("Graph*", ";", "WIN:1") //get list of unnamed open graphs
variable numgraphs = ItemsInList(graphlist, ";"), k = 0
if(KeepGraphsFlag == 0)
   for(k = 0; k<numgraphs; k+=1)
      KillWindow $StringFromList(k, graphlist, ";") //kill all unnamed open graphs
   endfor
   string killlist = DataWaveList() //list of all data waves (with name matching proper format)
   variable killlength = ItemsInList(killlist, ";")
   //kill all data waves not in use
   for(k=0; k<killlength; k+=1)
      //does not kill waves which are in graphs, tables, etc.
      KillWaves/Z $StringFromList(k, killlist, ";")
   endfor
elseif(KeepGraphsFlag == 1)
   string wlist = "" //list of waves in graph
   commandstr = ""
   variable numwaves = 0 //number of waves in graph
   variable waveappended = 0 //indicates if new wave could be appended to existing graph
else
   printf "KeepGraphsFlag value of %g invalid.  ", KeepGraphsFlag
```

```
   printf "Must be 0 or 1.  Aborting. (Vg1D)\r"
   Abort "Vg1D: invalid value of KeepGraphsFlag"
endif

variable numpts=numdivs+1
//update number of x and y points in ExperimentalDetails
FindValue/TEXT=("NumPoints X")/TXOP=4 ExperimentalDetailsName
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = num2istr(numpts)
else
   Print "\"NumPoints X\" not found in ExperimentalDetails."
   Print "Unable to record number of x-dimension points. (Vg1D)"
endif
FindValue/TEXT=("NumPoints Y")/TXOP=4 ExperimentalDetailsName
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = "0"
else
   Print "\"NumPoints Y\" not found in ExperimentalDetails."
   Print "Unable to record number of y-dimension points. (Vg1D)"
endif

variable nextwaveindex=nextwave(increment = 1) //increment data index by 1
wave/T datatypedmm
variable maxmeasurements = DimSize(datatypedmm, 0)
make/O/T/N=(maxmeasurements) wavenamelist = ""
string currentwavename = ""
variable starttime = 0, datatime = 0
variable n = 0, m = 0

make/O/N=(maxmeasurements, 2) vg1dSFs //wave to keep track of offsets and scale factors
variable scale, offset

string vg1dDeVars = "", recordvalue = "", dmmlipa = "", currentdatatype = ""
variable start = 0, stop = 0, actualrate = 0
variable/G Vg1Dnumused
//update Start, Stop, and Rate values in ExperimentalDetails
for(k=0; k<numVgs; k+=1)
   Vgstr = "Vg" + Vgnumlist[k]
   NVAR Vg1Dstart = $("V_start" + Vgstr)
   start += Vg1Dstart
   NVAR Vg1Dstop = $("V_stop" + Vgstr)
   stop += Vg1Dstop
   NVAR Vg1Drate = $("V_rate" + Vgstr)
   actualrate = Vg1Drate * Vg1Dnumused
   FindValue/TEXT=(Vgstr + "Start")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      sprintf recordvalue, "%g", Vg1Dstart
      ExperimentalDetails[V_value][1] = recordvalue
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", Vgstr + "Start"
      printf "Unable to record start value. (Vg1D)\r"
   endif
   FindValue/TEXT=(Vgstr + "Stop")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      sprintf recordvalue, "%g", Vg1Dstop
      ExperimentalDetails[V_value][1] = recordvalue
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", Vgstr + "Stop"
      printf "Unable to record stop value. (Vg1D)\r"
   endif
   FindValue/TEXT=(Vgstr + "Rate")/TXOP=4 ExperimentalDetailsName
   if(V_Value > -1)
      sprintf recordvalue, "%g", actualrate
```

```
      ExperimentalDetails[V_value][1] = recordvalue
   else
      printf "\"%s\" not found in ExperimentalDetails.  ", Vgstr + "Rate"
      printf "Unable to record sweep rate. (Vg1D)\r"
   endif
endfor

variable windowleft, windowtop, windowright, windowbottom
for(n=0; n < maxmeasurements; n+=1) //step though each dependent variable to be measured
   currentdatatype = datatypedmm[n][0]
   if(!stringmatch(currentdatatype,"-"))
      scale = 1 //for ReadDMM returning V
      offset = 0

      for(m=0; m<maxmeasurements; m+=1)
         if((m != n) && (stringmatch(currentdatatype, datatypedmm[m][0])))
            printf "Dependent variable %s chosen more than once ", currentdatatype
            printf "and will be overwritten.  Aborting. (Vg1D)\r"
            Abort "Vg1D: dependent variable chosen multiple times."
         endif
      endfor

      //get dmm offset (y-intercept)
      FindValue/TEXT=("DMMoffset" + datatypedmm[n][2])/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         offset += str2num(ExperimentalDetails[V_value][1])
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", "DMMoffset" + datatypedmm[n][2]
         printf "Offset not changed. (Vg1D)\r"
      endif
      //get dmm scale factor
      FindValue/TEXT=("DMMscale" + datatypedmm[n][2])/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         scale *= str2num(ExperimentalDetails[V_Value][1])
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", "DMMscale" + datatypedmm[n][2]
         printf "Scale not changed. (Vg1D)\r"
      endif
      if(!stringmatch(datatypedmm[n][4], "0"))
         //get lock-in sensitivity
         FindValue/TEXT=("LIsens" + datatypedmm[n][4])/TXOP=4 ExperimentalDetailsName
         if(V_Value > -1)
            scale *= str2num(ExperimentalDetails[V_value][1])/10
         else
            printf "\"%s\" not found in ExperimentalDetails.  ", "LIsens" + datatypedmm[n][3]
            printf "Scale not changed. (Vg1D)\r"
         endif
      endif
      if(!stringmatch(datatypedmm[n][5], "0"))
         //get pre-amp amplification
         FindValue/TEXT=("PAamp" + datatypedmm[n][5])/TXOP=4 ExperimentalDetailsName
         if(V_Value > -1)
            scale /= str2num(ExperimentalDetails[V_value][1])
         else
            printf "\"%s\" not found in ExperimentalDetails.  ", "PAamp" + datatypedmm[n][4]
            printf "Scale not changed. (Vg1D)\r"
         endif
      endif

      scale *= ScaleFactor(n)

      //update vg1dSFs wave
      vg1dSFs[n][0] = offset
```

```
      vg1dSFs[n][1] = scale

      vg1dDeVars += currentdatatype+ ";" //keep track of dependent variables to be measured
      //record DMM, lock-in, preamp chain
      dmmlipa += datatypedmm[n][2] + datatypedmm[n][4] + datatypedmm[n][5] + ";"

      sprintf currentwavename, "%s_%s_%d", datatypedmm[n][0], idstr, nextwaveindex
      wavenamelist[n] = currentwavename //keep list of wave names
      make/O/N=(numpts) $currentwavename = NaN //make the wave
      if(start != stop)
         SetScale/I x start, stop, $currentwavename //set x scaling
      endif
      if(KeepGraphsFlag == 0) //make new graphs for next set of data
         Display $currentwavename //create graph
         Label bottom, idstr + " (" + ivarunits + ")" //label x-axis
         Label left, datatypedmm[n][0] + " (" + datatypedmm[n][1] +")" //label y-axis
         GraphStyle() //edit function to change graph style
         //record DMM, lock-in, preamp chain
         movewindow/I (mod(n, 3)*3), (floor(n/3)*3), (mod(n, 3)*3 + 2.95), (floor(n/3)*3 + 2.2)
      elseif(KeepGraphsFlag == 1) //display data on old graphs
         waveappended = 0
         for(m=0; m<numgraphs; m+=1) //check each graph for match of devar
            //writes list of waves in graph to W_WaveList
            commandstr = "GetWindow " + StringFromList(m, graphlist, ";") + ", wavelist"
            Execute/Q commandstr
            wlist = TWave2Str(W_WaveList) //generate wlist
            numwaves = ItemsInList(wlist, ";") //number of waves on graph
            for(k=0; k<numwaves; k+=1) //check each wave to see if match devar
               //if match, append new trace
               if(stringmatch(StringFromList(k, wlist, ";"), currentdatatype + "_*"))
                  commandstr = "AppendToGraph/W=" + StringFromList(m, graphlist, ";")
                  commandstr += " " + currentwavename
                  Execute/Q commandstr
                  waveappended = 1
                  break
               endif
            endfor
         endfor
         if(waveappended == 0)
            Display $currentwavename //create graph
            Label bottom, idstr + " (" + ivarunits + ")" //label x-axis
            Label left, datatypedmm[n][0] + " (" + datatypedmm[n][1] +")" //label y-axis
            GraphStyle() //edit function to change graph style
            windowleft = (mod(n+numgraphs, 3)*3)
            windowtop = (floor((n+numgraphs)/3)*3)
            windowright = (mod(n+numgraphs, 3)*3 + 2.95)
            windowbottom = (floor((n+numgraphs)/3)*3 + 2.2)
            movewindow/I windowleft, windowtop, windowright, windowbottom //move graph window
         endif
      endif
   endif
endfor
//update dependent variable in ExperimentalDetails
FindValue/TEXT=("Dependent")/TXOP=4 ExperimentalDetailsName
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = vg1dDeVars
else
   Print "\"Dependent\" not found in ExperimentalDetails."
   Print "Unable to record dependent variables. (Vg1D)"
endif
//update dmm/lock-in/preamp info in ExperimentalDetails
FindValue/TEXT=("DMMLIPA")/TXOP=4 ExperimentalDetailsName
if(V_Value > -1)
```

```
   ExperimentalDetails[V_value][1] = dmmlipa
else
   Print "\"DMMLIPA\" not found in ExperimentalDetails."
   Print "Unable to record dependent variables. (Vg1D)"
endif

FindValue/TEXT=("Date")/TXOP=4 ExperimentalDetailsName //update date
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = date()
else
   Print "\"Date\" not found in ExperimentalDetails.  Unable to record date. (Vg1D)"
endif
FindValue/TEXT=("StartTime")/TXOP=4 ExperimentalDetailsName //update start time
if(V_Value > -1)
   ExperimentalDetails[V_value][1] = time()
else
   Print "\"StartTime\" not found in ExperimentalDetails.  Unable to record time. (Vg1D)"
endif

//save ExperimentalDetails to file
Duplicate/O/T/R=[][1] ExperimentalDetails vg1dDetails
ExperimentalDetails[][DimSize(ExperimentalDetails, 1)-1] = vg1dDetails[p]
Concatenate/T {ExperimentalDetailsName}, ExperimentalDetails
SaveDetails()

//constants for mixing chamber temperature fit
variable A0 = 411.7925271369486, A1 = -487.3966692536534, A2 = 233.1118681633504
variable A3 = -55.85454979733308, A4 = 6.690672513604219, A5 = -0.3205752792140923

make/O/N=(numpts,numVgs) vg1dInVals //wave of independent variable points
make/O/N=(numVgs) vg1dRates
for(k=0; k<numVgs; k+=1)
   Vgstr = "Vg" + Vgnumlist[k]
   NVAR Vg1Dstart = $("V_start" + Vgstr)
   NVAR Vg1Dstop = $("V_stop" + Vgstr)
   NVAR Vg1Drate = $("V_rate" + Vgstr)
   vg1dInVals[][k]=Vg1Dstart+p*(Vg1Dstop-Vg1Dstart)/numdivs
   vg1dRates[k] = Vg1Drate * Vg1Dnumused
endfor

// data taking loop
make/O/N=(maxmeasurements) vg1dcurrentvals = NaN
make/O vg1dcurrentwave
for(k=0; k<numVgs; k+=1)
   Vgstr = "Vg" + Vgnumlist[k]
   SweepVal(Vgstr, vg1dInVals[0][k], Vg1Drates[k], 0)
endfor
//wait(1)
try
   starttime = stopMSTimer(-2)
   for(m=0; m<numpts; m+=1)
      for(k=0; k<numVgs; k+=1)
         Vgstr = "Vg" + Vgnumlist[k]
         SweepVal(Vgstr, vg1dInVals[m][k], Vg1Drates[k], 0) //set independent variable
      endfor
      TakeData(vg1dcurrentvals) //record data from dmms
      for(n=0; n<maxmeasurements; n+=1) //step through waves being taken
         if(!stringmatch(datatypedmm[n][0], "-"))
            currentwavename = wavenamelist[n]
            wave vg1dcurrentwave = $currentwavename
            //update wave with with scaled data
            vg1dcurrentwave[m] = (vg1dcurrentvals[n]-vg1dSFs[n][0])*vg1dSFs[n][1]
            if(stringmatch(datatypedmm[n][0], "Tmc")) //implement nonlinear temperature fit
```

```
                vg1dcurrentwave[m] = A0 + A1 * log(vg1dcurrentwave[m])
                vg1dcurrentwave[m] += A2*(log(vg1dcurrentwave[m]))^2
                vg1dcurrentwave[m] += A3*(log(vg1dcurrentwave[m]))^3
                vg1dcurrentwave[m] += A4*(log(vg1dcurrentwave[m]))^4
                vg1dcurrentwave[m] += A5*(log(vg1dcurrentwave[m]))^5
             vg1dcurrentwave[m] = 10^vg1dcurrentwave[m]
             endif
          endif
       endfor
       DoUpdate //Comment this line if no data will be displayed during acquisition JBM060828
    endfor
    datatime = stopMSTimer(-2)
 catch
    datatime = stopMSTimer(-2)
    DoUpdate
 endtry
 AllDMMLocal() //do we want this?
 try //save waves and update ExperimentalDetails even if function aborted
    for(n=0; n<maxmeasurements; n+=1)
       if(!stringmatch(datatypedmm[n][0], "-")) //save waves as Igor binary files
          currentwavename = wavenamelist[n]
          Save/P=binarypath $currentwavename
          Save/P=backuppath $currentwavename
       endif
    endfor

    //update Start, Stop, and Rate to current values (may not match input if function was aborted)
    for(k=0; k<numVgs; k+=1)
       Vgstr = "Vg" + Vgnumlist[k]
       NVAR Vg1Dstart = $("V_start" + Vgstr)
       FindValue/TEXT=(Vgstr + "Start")/TXOP=4 ExperimentalDetailsName
       if(V_Value > -1)
          sprintf recordvalue, "%g", vg1dInVals[m][k]
          ExperimentalDetails[V_value][1] = recordvalue
       else
          printf "Warning: unable to update %s after sweep. (Vg1D)\r", Vgstr + "Start"
       endif
       FindValue/TEXT=(Vgstr + "Stop")/TXOP=4 ExperimentalDetailsName
       if(V_Value > -1)
          sprintf recordvalue, "%g", vg1dInVals[m][k]
          ExperimentalDetails[V_value][1] = recordvalue
       else
          printf "Warning: unable to update %s after sweep. (Vg1D)\r", Vgstr+ "Stop"
       endif
       FindValue/TEXT=(Vgstr + "Rate")/TXOP=4 ExperimentalDetailsName
       if(V_Value > -1)
          ExperimentalDetails[V_value][1] = "0"
       else
          printf "Warning: unable to update %s after sweep. (Vg1D)\r", Vgstr + "Rate"
       endif
    endfor
 catch
    Print "Warning: function aborted, data may not be saved."
    Print "Check that all files are properly saved. (Vg1D)"
 endtry
 printf "Created waves #%d, finished at %s, ",nextwaveindex,time(),
 printf "elapsed time %6.3f min.\r"((datatime-starttime)*1e-6/60)
 sprintf recordstr, "Created waves #%d, finished at %s, ",nextwaveindex,time()
 sprintf recordstr2, "elapsed time %6.3f min.\r",((datatime-starttime)*1e-6/60)
 recordstr += recordstr2
 Notebook Record, selection={StartofFile, StartofFile}, text = recordstr
 SaveNotebook/O/P=savepath/S=6 Record as "Record.txt"
end
```

```
function Vg1DCancel(ctrlName) : ButtonControl
   string ctrlName

   DoWindow/K Vg1DPanel
   Abort "Vg1D: user cancel."
end

function Vg1DValidate(ctrlName) : ButtonControl
   string ctrlName

   wave Vg1DUse
   variable n = 0
   variable actualrate = 0
   ControlInfo $("doRun")
   if(V_disable == 0)
      Button doRun, disable=2
      for(n=0; n<DimSize(Vg1DUse, 0); n+=1)
         ControlInfo $("UseVg" + num2istr(n))
         if(V_flag != 0)
            CheckBox $("UseVg" + num2istr(n)), disable=0
         endif
         if(Vg1DUse[n] == 1)
            SetVariable $("StartVg" + num2istr(n)), disable=0
            SetVariable $("StopVg" + num2istr(n)), disable=0
            SetVariable $("RateVg" + num2istr(n)), disable=0
         endif
      endfor
   else
      wave/T ExperimentalDetails, ExperimentalDetailsName
      variable maxrate = 0, okRun = 1, divider = Inf, hardmax = 0, expmax = 0, hardmin = 0, expmin = 0
      variable/G Vg1Dnumused = 0
      make/o/n=0 Vg1DValidateTimes
      for(n=0; n<DimSize(Vg1DUse, 0); n+=1)
         if(Vg1DUse[n] == 1)
            Vg1Dnumused += 1
         endif
      endfor
      for(n=0; n<DimSize(Vg1DUse, 0); n+=1)
         if(Vg1DUse[n] == 1)
            NVAR rate = $("V_rateVg" + num2istr(n))
            actualrate = rate * Vg1Dnumused
            if((rate == 0) || stringmatch("nan", num2str(rate)))
               SetVariable $("rateVg" + num2istr(n)), valueBackColor=(65535,0,0)
               okRun = 0
            endif
            FindValue/TEXT=("Vg" + num2istr(n) + "RateMax")/TXOP=4 ExperimentalDetailsName
            if(V_Value > -1)
               maxrate =  str2num(ExperimentalDetails[V_value][1])
            else
               printf "\"%s\" not found in ExperimentalDetails.  ", "Vg" + num2istr(n) + "RateMax"
               printf "Sweep rate may be too fast. (Vg1DValidate)\r"
               SetVariable $("RateVg" + num2istr(n)), valueBackColor=(65535,0,0)
               break
            endif
            if(actualrate > maxrate)
               printf "Vg%d actual rate too large.  ", n, maxrate
               printf "Must be no more than %g mV/sec. (Vg1DValidate)\r"
               SetVariable $("RateVg" + num2istr(n)), valueBackColor=(65535,0,0)
               okRun = 0
            endif
            NVAR start = $("V_startVg" + num2istr(n))
            NVAR stop = $("V_stopVg" + num2istr(n))
```

```
            InsertPoints 0, 1, Vg1DValidateTimes
            Vg1DValidateTimes[0] = abs(start - stop)/rate
         endif
      endfor

      if(Vg1Dnumused<2)
         Print "Must use at least two gates. (Vg1DValidate)"
         okRun = 0
      else
         WaveStats/Q Vg1DValidateTimes
         if(V_sdev !=0)
            Print "Rates do not match.  Please update a rate. (Vg1DValidate)"
            okRun = 0
         endif
      endif

      if(okRun == 1)
         for(n=0; n<DimSize(Vg1DUse, 0); n+=1)
            ControlInfo $("UseVg" + num2istr(n))
            if(V_flag != 0)
               CheckBox $("UseVg" + num2istr(n)), disable=2
            endif
            if(Vg1DUse[n] == 1)
               SetVariable $("StartVg" + num2istr(n)), disable=2
               SetVariable $("StopVg" + num2istr(n)), disable=2
               SetVariable $("RateVg" + num2istr(n)), disable=2
            endif
         endfor
         Button doRun, disable=0
      endif
   endif
end

function Vg1DRun(ctrlName) : ButtonControl
   string ctrlName

   DoWindow/K Vg1DPanel
end

function Vg1DCheck(ctrlName, checked) : CheckBoxControl
   string ctrlName
   variable checked

   wave Vg1DUse
   wave/T ExperimentalDetails, ExperimentalDetailsName
   string Vgname = ctrlname[3, 5]
   variable Vgnum = str2num(Vgname[2])
   variable divider = Inf, hardmin = 0, hardmax = 0
   variable Vgmin = 0, Vgmax = 0, Ratemax = 0
   string commandstr1 = "", commandstr2 = "", commandstr3 = ""
   if(checked == 1)
      FindValue/TEXT=(Vgname + "Divider")/TXOP=4 ExperimentalDetailsName //get divider value
      if(V_Value > -1)
         if(str2num(ExperimentalDetails[V_value][1]) > 0)
            divider = str2num(ExperimentalDetails[V_value][1])
         else
            printf "\"%s\" value not positive.  Aborting. (Vg1DCheck)\r", Vgname + "Divider"
            Abort "Vg1DCheck: Vg#Divider not positive."
         endif
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", Vgname + "Divider"
         printf "Aborting. (Vg1DCheck)\r"
         Abort "Vg1DCheck: cannot find Vg#Divider in ExperimentalDetails."
```

```
    endif
    FindValue/TEXT=(Vgname + "ExpMin")/TXOP=4 ExperimentalDetailsName
    if(V_Value > -1)
       Vgmin = str2num(ExperimentalDetails[V_value][1])
    else
       printf "\"%s\" not found in ExperimentalDetails.  ", Vgname + "ExpMin"
       printf "Unable to check if value exceeds minimum. (Vg1DCheck)\r"
       Vgmin = -Inf
    endif
    FindValue/TEXT=(Vgname + "HardMin")/TXOP=4 ExperimentalDetailsName
    if(V_Value > -1)
       hardmin = str2num(ExperimentalDetails[V_value][1])
    else
       printf "\"%s\" not found in ExperimentalDetails.  ", Vgname + "ExpMin"
       printf "Unable to check if value exceeds minimum. (Vg1DCheck)\r"
       hardmin = -Inf
    endif
    if(hardmin > Vgmin)
       Vgmin = hardmin
    endif

    FindValue/TEXT=(Vgname + "ExpMax")/TXOP=4 ExperimentalDetailsName
    if(V_Value > -1)
       Vgmax = str2num(ExperimentalDetails[V_value][1])
    else
       printf "\"%s\" not found in ExperimentalDetails.  ", Vgname + "ExpMax"
       printf "Unable to check if value exceeds maximum. (Vg1DCheck)\r"
       Vgmax = Inf
    endif
    FindValue/TEXT=(Vgname + "HardMax")/TXOP=4 ExperimentalDetailsName
    if(V_Value > -1)
       hardmax = str2num(ExperimentalDetails[V_value][1])
    else
       printf "\"%s\" not found in ExperimentalDetails.  ", Vgname + "ExpMin"
       printf "Unable to check if value exceeds minimum. (Vg1DCheck)\r"
       hardmax = Inf
    endif
    if(hardmax < Vgmax)
       Vgmax = hardmax
    endif

    FindValue/TEXT=(Vgname + "RateMax")/TXOP=4 ExperimentalDetailsName
    if(V_Value > -1)
       Ratemax = str2num(ExperimentalDetails[V_value][1])
    else
       printf "\"%s\" not found in ExperimentalDetails.  ", Vgname + "RateMax"
       printf "Sweep rate may be too fast. (Vg1DCheck)\r"
       Ratemax = Inf
    endif
    sprintf commandstr1 "SetVariable start%s, limits={%g,%g,0}, disable=0", Vgname, Vgmin, Vgmax
    sprintf commandstr2 "SetVariable stop%s, limits={%g,%g,0}, disable=0", Vgname, Vgmin, Vgmax
    sprintf commandstr3 "SetVariable rate%s, limits={0,%g,0}, disable=0", Vgname, Ratemax
    Execute/Q commandstr1
    Execute/Q commandstr2
    Execute/Q commandstr3
    Vg1DUse[Vgnum] = 1
 else
    sprintf commandstr1 "SetVariable start%s, disable=1", Vgname
    sprintf commandstr2 "SetVariable stop%s, disable=1", Vgname
    sprintf commandstr3 "SetVariable rate%s, disable=1", Vgname
    Execute/Q commandstr1
    Execute/Q commandstr2
    Execute/Q commandstr3
```

```
      Vg1DUse[Vgnum] = 0
   endif
end

function Vg1DRate(ctrlname, varNum, varStr, varName) : SetVariableControl
   string ctrlName
   variable varNum
   string varStr, varName

   wave Vg1DUse
   variable n=0
   string Vgname = ctrlname[4, 6]
   NVAR Vgstart = $("V_start" + Vgname)
   NVAR Vgstop = $("V_stop" + Vgname)
   for(n=0; n<DimSize(Vg1DUse, 0); n+=1)
      ControlInfo $("RateVg" + num2istr(n))
      if(V_flag != 0)
         SetVariable $("RateVg" + num2istr(n)), valueBackColor=(65535,65535,65535)
      endif
      if(n != str2num(ctrlname[6]))
         NVAR rate = $("V_rateVg" + num2istr(n))
         NVAR start = $("V_startVg" + num2istr(n))
         NVAR stop = $("V_stopVg" + num2istr(n))
         rate = varNum * abs(stop - start) / abs(Vgstop - Vgstart)
      endif
   endfor
end

//panel to choose dependent variables to measure
//with corresponding dmm, lock-in, and pre-amp specifications
function datatypes([numcontrols]) : Panel
   variable numcontrols //maximum number of simultaneous dependent variables
   if(ParamIsDefault(numcontrols))
      numcontrols = 5
   endif
   variable n = 0, yposition = 0
   string commandstr0 = "", commandstr = "", commandstr2 = "", commandstr3 = ""
   string commandstr4 = "", commandstr5 = "", commandstr6 = "", commandstr7 = "", commandstr8 = ""
   string commandstrb = ""
   make/O/T/N=(numcontrols, 6) datatypedmm = "-"
   datatypedmm[][2,5] = "0"
   PauseUpdate; Silent 1      // building window...
   NewPanel/W=(1176,50,1432,50+18+18*numcontrols) as "Data Types" //create window
   UpdateVariableLists()
   sprintf commandstr0 "CheckBox samegraph,pos={9,0},size={14,14},"
   commandstr0 += "proc=SameGraphCheck,title=\"Keep Graphs\""
   Execute/Q commandstr0
   for(n=1; n<numcontrols+1; n+=1) //create set of controls for each possible variable
      yposition = n * 17
      sprintf commandstr0, "variable/G V_dmmnum%d = 0, V_lockinnum%d = 0, V_ampnum%d = 0", n, n, n
      sprintf commandstr, "PopupMenu datatype%d,pos={9,%d},", n, yposition
      sprintf commandstrb, "size={61,21},proc=setdatatype,title=\"%d\"", n
      commandstr += commandstrb
      sprintf commandstr2, "PopupMenu datatype%d,mode=1,bodyWidth= 50,", n
      commandstr2 += "popvalue=\"-\",value= #\"DeVarList\""
      sprintf commandstr3, "SetVariable setdmm%d,pos={70,%d},size={70,16},", n, (yposition+1)
      commandstr3 += "disable=1,proc=setdmmnum,title=\"dmm\""
      sprintf commandstr4, "SetVariable setdmm%d,limits={0,31,1},value= V_dmmnum%d", n, n
      sprintf commandstr5, "SetVariable setlockin%d,pos={141,%d},size={50,16},", n, (yposition+1)
      commandstr5 += "disable=1,proc=setlockinnum,title=\"LI\""
      sprintf commandstr6, "SetVariable setlockin%d,limits={0,31,1},value= V_lockinnum%d", n, n
      sprintf commandstr7, "SetVariable setamp%d,pos={192,%d},size={65,16},", n, (yposition+1)
      commandstr7 += "disable=1,proc=setampnum,title=\"amp\""
```

```
      sprintf commandstr8, "SetVariable setamp%d,limits={-31,31,1},value= V_ampnum%d", n, n
      Execute/Q commandstr0
      Execute/Q commandstr
      Execute/Q commandstr2
      Execute/Q commandstr3
      Execute/Q commandstr4
      Execute/Q commandstr5
      Execute/Q commandstr6
      Execute/Q commandstr7
      Execute/Q commandstr8
   endfor
end

//controls whether to add new data to old graphs or make new graphs
function SameGraphCheck(ctrlName, checked) : CheckBoxControl
   string ctrlName
   variable checked
   NVAR KeepGraphsFlag

   if(checked == 1)
      KeepGraphsFlag = 1
   else
      KeepGraphsFlag = 0
   endif
end

//update datatypedmm wave with chosen dependent variable
Function setdatatype (ctrlName,popNum,popStr) : PopupMenuControl
   //also control visibility of dmm, lock-in, and pre-amp controls
   String ctrlName
   Variable popNum   // which item is currently selected (1-based)
   String popStr     // contents of current popup item as string
   wave/T datatypedmm
   variable ctrlNum = str2num(ctrlName[8])
   wave/T DeVarWave
   FindValue/TEXT=popStr/TXOP=4 DeVarWave
   datatypedmm[ctrlNum-1][0] = popStr //dependent variable
   datatypedmm[ctrlNum-1][1] = DeVarWave[V_value][1] //units
   string setdmmname = "", setlockinname = "", setampname = ""
   sprintf setdmmname,"setdmm%s",ctrlname[8]
   sprintf setlockinname, "setlockin%s", ctrlname[8]
   sprintf setampname, "setamp%s", ctrlname[8]
   //disable dmm, lock-in, and pre-amp controls if no dependent variable chosen
   setvariable $setdmmname,disable=(stringmatch(popstr,"-"))
   setvariable $setlockinname,disable=(stringmatch(popstr,"-"))
   setvariable $setampname,disable=(stringmatch(popstr,"-"))
end

//updates datatypedmm with dmm number chosen
Function setdmmnum(ctrlName,varNum,varStr,varName) : SetVariableControl
   String ctrlName
   Variable varNum
   String varStr
   String varName

   string DMMstring = num2istr(varNum)
   wave/T ExperimentalDetails
   Duplicate/O/R=[][0] ExperimentalDetails, ExperimentalDetailsName
   //check for DMMoffset and DMMscale values; print warning if not defined
   if(varNum != 0)
      FindValue/TEXT=("DMMoffset" + DMMstring)/TXOP=4 ExperimentalDetailsName
      if(V_value == -1)
         printf "Warning: no zero offset defined for DMM%s. (SetDMMNum)\r", DMMstring
```

```
      elseif(StringMatch(num2str(str2num(ExperimentalDetails[V_value][1])),"NaN"))
         printf "Warning: invalid zero offset defined for DMM%s. (SetDMMNum)\r", DMMstring
      endif
      FindValue/TEXT=("DMMscale" + DMMstring)/TXOP=4 ExperimentalDetailsName
      if(V_value == -1)
         printf "Warning: no scale factor defined for DMM%s. (SetDMMNum)\r", DMMstring
      elseif(StringMatch(num2str(str2num(ExperimentalDetails[V_value][1])),"NaN"))
         printf "Warning: invalid scale factor defined for DMM%s. (SetDMMNum)\r", DMMstring
      endif
   endif

   string dmmname,dmmvarname
   wave/T datatypedmm
   variable ctrlNum = str2num(ctrlName[6])
   sprintf dmmname,"dmm%d",varNum
   sprintf dmmvarname,"V_dmm%s",ctrlname[6]
   NVAR dmm=$dmmname
   if(NVAR_exists(dmm)) //if dmm found update datatypedmm
      datatypedmm[ctrlNum-1][3] = num2istr(dmm)
      setvariable $ctrlname,labelback=0
   else //if dmm not found, set value to 0 and highlight in red
      datatypedmm[ctrlNum-1][3] = "0"
      setvariable $ctrlname,labelback=(65535,0,0)
   endif
   datatypedmm[ctrlNum-1][2] = num2istr(varNum)
end

//update datatypedmm with lock-in number
function setlockinnum(ctrlName, varNum, varStr, varName) : SetVariableControl
   string ctrlName, varStr, varName
   variable varNum

   string LIstring = num2istr(varNum)
   wave/T ExperimentalDetails
   if(varNum != 0)
      setvariable $ctrlname,labelback=0
      Duplicate/O/R=[][0] ExperimentalDetails, ExperimentalDetailsName
      //check for lock-in sensitivity and time constant; print warning if not defined
      FindValue/TEXT=("LIsens" + LIstring)/TXOP=4 ExperimentalDetailsName
      if(V_value == -1)
         printf "Warning: no lock-in sensitivity defined for LI%s. (SetLockInNum)\r", LIstring
         setvariable $ctrlname,labelback=(65535,0,0)
      elseif(StringMatch(num2str(str2num(ExperimentalDetails[V_value][1])),"NaN"))
         printf "Warning: invalid lock-in sensitivity defined for LI%s. (SetLockInNum)\r", LIstring
         setvariable $ctrlname,labelback=(65535,0,0)
      endif
      Duplicate/O/R=[][0] ExperimentalDetails, ExperimentalDetailsName
      FindValue/TEXT=("LItau" + LIstring)/TXOP=4 ExperimentalDetailsName
      if(V_value == -1)
         printf "Warning: no lock-in time constant defined for LI%s. (SetLockInNum)\r", LIstring
         setvariable $ctrlname,labelback=(65535,0,0)
      elseif(StringMatch(num2str(str2num(ExperimentalDetails[V_value][1])),"NaN"))
         printf "Warning: invalid lock-in time constant defined for LI%s. (SetLockInNum)\r", LIstring
         setvariable $ctrlname,labelback=(65535,0,0)
      endif
   else
      setvariable $ctrlname,labelback=(65535,0,0)
   endif

   wave/T datatypedmm
   variable ctrlNum = str2num(ctrlName[9])
   datatypedmm[ctrlNum-1][4] = LIstring //update datatypedmm
end
```

```
//update datatypedmm with pre-amp number
function setampnum(ctrlName, varNum, varStr, varName) : SetVariableControl
   string ctrlName, varStr, varName
   variable varNum

   string PAstring = num2istr(varNum)
   wave/T ExperimentalDetails
   if(varNum != 0)
      setvariable $ctrlname,labelback=0
      Duplicate/O/R=[][0] ExperimentalDetails, ExperimentalDetailsName
      //check for pre-amp amplification; print warning if not defined
      FindValue/TEXT=("PAamp" + PAstring)/TXOP=4 ExperimentalDetailsName
      if(V_value == -1)
         printf "Warning: no amplification defined for PA%s. (SetAmpNum)\r", PAstring
         setvariable $ctrlname,labelback=(65535,0,0)
      elseif(StringMatch(num2str(str2num(ExperimentalDetails[V_value][1])),"NaN"))
         printf "Warning: invalid amplification defined for PA%s. (SetAmpNum)\r", PAstring
         setvariable $ctrlname,labelback=(65535,0,0)
      endif
   else
      setvariable $ctrlname,labelback=(65535,0,0)
   endif

   wave/T datatypedmm
   variable ctrlNum = str2num(ctrlName[6])
   datatypedmm[ctrlNum-1][5] = num2istr(VarNum) //update datatypedmm
end

function SaveDetails() // Save the ExperimentalDetails to file after a scan is done
   wave/T ExperimentalDetails
   string savename = "Details" + num2istr(nextwave() - 1)

   Duplicate/T/O/R=[][0,2] ExperimentalDetails $savename
   Duplicate/T/O/R=[][DimSize(ExperimentalDetails, 1)-2] ExperimentalDetails SaveDetailsWave
   Concatenate/T {SaveDetailsWave}, $savename

   //edit to switch x and y axes
   //general text formal for Excel
   Save/G/O/W/P=savepath ExperimentalDetails as "ExperimentalDetails.xls"
   Save/O/C/P=savepath ExperimentalDetails as "ExperimentalDetails.ibw" //Igor binary
   //individual wave details saved as backup; Igor binary
   Save/C/P=backuppath $savename as savename + ".ibw"

   SaveNotebook/O/P=savepath/S=1 Record as "Record.txt"
end

//Load the waves with indices starting at LoadedIndex + 1
Function Loadibw()
   variable/G LoadedIndex

   string ibwList = IndexedFile(binarypath, -1, ".ibw")
   string indexList = "", currentFile = "", devar = "", invar1 = "", invar2 = ""
   string devarunits = "", invar1units = "", invar2units = ""
   LoadWave/O/H/P=savepath "ExperimentalDetails.ibw" //load ExperimentalDetails
   Duplicate/O/R=[][0] ExperimentalDetails, ExperimentalDetailsName
   variable LastIndex = nextwave() - 1 //last index to check equal to index of last wave taken
   variable n = 0, m = 0, k = 0
   //update InVarExp and DeVarExp to match current independent and dependent variable lists
   UpdateVariableLists()
   SVAR DeVarExp, InVarExp
   //regex expression to match names
   string matchexp = "(?i)((" + DeVarExp + ")(_(" + InVarExp + )){1,2}_"
```

```
WAVE/T DeVarWave, InVarWave
string leftlabel = "", rightlabel = ""
string currentmatch = ""

string cscalelegend = ""
variable windowleft, windowtop, windowright, windowbottom
for(n = LoadedIndex + 1; n < LastIndex + 1; n+=1) //try to load and graph each index
   LoadedIndex = n
   currentmatch = matchexp + num2istr(n) + ".ibw)"
   indexList = GrepList(ibwList, currentmatch) //check for valid file name
   if(strlen(indexlist) > 0)
      indexList = SortList(indexList, ";", 0) //sort alphabetically
      currentFile = StringFromList(0, indexList, ";")
      currentFile = RemoveEnding(currentFile, ".ibw")
      if(stringmatch(currentFile, "*_*_*_" + num2istr(n))) //for 2D waves
         //get independent variables and units
         invar1 = StringFromList(1, currentFile, "_")
         invar1=UpperStr(invar1[0]) + invar1[1, Inf]
         if(StringMatch(invar1[0,1], "Vg"))
            FindValue/TEXT=("Vg")/TXOP=4 InVarWave
         else
            FindValue/TEXT=(invar1)/TXOP=4 InVarWave
         endif
         invar1units = InVarWave[V_value][1]
         invar2 = StringFromList(2, currentFile, "_")
         invar2=UpperStr(invar2[0]) + invar2[1, Inf]
         if(StringMatch(invar2[0,1], "Vg"))
            FindValue/TEXT=("Vg")/TXOP=4 InVarWave
         else
            FindValue/TEXT=(invar2)/TXOP=4 InVarWave
         endif
         invar2units = InVarWave[V_value][1]
         for(m=0; m<ItemsInlist(indexList, ";"); m+=1) //load and graph each wave
            currentFile = StringFromList(m, indexList, ";")
            LoadWave/H/P=binarypath currentFile //load wave
            currentFile = RemoveEnding(currentFile, ".ibw")
            SetWaveLock 1, $currentFile
            //get dependent variable and units
            devar = StringFromList(0, currentFile, "_")
            FindValue/TEXT=(devar)/TXOP=4 DeVarWave
            devarunits = DeVarWave[V_value][1]
            display; appendimage $currentFile //graph
            ModifyImage $currentFile ctab= {*,*,Terrain256,0} //set color scheme
            GraphStyle2D()
            //create color scale
            ColorScale/C/N=cslegend/B=1/F=0/A=RC/X=0.00/Y=5.00/E=2 image = $currentFile
            ColorScale/C/N=cslegend lblMargin=0
            cscalelegend = devar + StringFromList(3, currentFile, "_") + " (" + devarunits + ")"
            ColorScale/C/N=cslegend cscalelegend
            Label left, invar1 + " (" + invar1units + ")" //label y-axis
            Label bottom, invar2 + " (" + invar2units + ")" //label x-axis
            //label color scale
            //move graph window
            movewindow/I (mod(m, 3)*3), (floor(m/3)*3), (mod(m, 3)*3 + 2.95), (floor(m/3)*3 + 2.2)
            DoWindow/B kwTopWin //put window on bottom
         endfor

      elseif(stringmatch(currentFile, "*_*_" + num2istr(n))) //for 1D waves
         //get independent variable and units
         invar1 = StringFromList(1, currentFile, "_")
         invar1=UpperStr(invar1[0]) + invar1[1, Inf]
         if(StringMatch(invar1[0,1], "Vg"))
            FindValue/TEXT=("Vg")/TXOP=4 InVarWave
```

```
else
   FindValue/TEXT=(invar1)/TXOP=4 InVarWave
endif
invar1units = InVarWave[V_value][1]
for(m=0; m<ItemsInlist(indexList, ";"); m+=1) //load and graph waves
   currentFile = StringFromList(m, indexList, ";")
   LoadWave/H/P=binarypath currentFile //load wave
   //get dependent variable and units
   currentFile = RemoveEnding(currentFile, ".ibw")
   SetWaveLock 1, $currentFile
   devar = StringFromList(0, currentFile, "_")
   devar=UpperStr(devar[0]) + devar[1, Inf]
   FindValue/TEXT=(devar)/TXOP=4 DeVarWave
   devarunits = DeVarWave[V_value][1]
   strswitch(devar) //check if dependent variable is a resistance
      case "Rd":
      case "Rdp":
      case "Rdm":
      case "RdDC":
      case "Rl":
      case "Rxx":
      case "RxxDC":
      case "Rxy":
      case "RxyDC":
         //plot Rl and Rxx on left axis
         if(stringmatch(devar, "Rl") || stringmatch(devar[0,2], "Rxx"))
            display $currentFile
            ModifyGraph axRGB(left)=(65280,0,0)
            leftlabel = devar
            Label bottom, invar1 + " (" + invar1units + ")"
         else //plot Rd* and Rxy on right axis
            display/R $currentFile
            if(StringMatch(devar[0,1], "Rd"))
               ModifyGraph rgb($currentFile) = (0, 0, 65280)
            else
               ModifyGraph rgb($currentFile) = (0, 0, 0)
            endif
            rightlabel = devar
            Label bottom, invar1 + " (" + invar1units + ")"
         endif
         indexList = RemoveFromList(currentFile + ".ibw", indexList, ";")
         m -= 1
         GraphStyle()
         //check for any additional resistance waves and remove them from the list
         for(k=m+1; k<ItemsInlist(indexList, ";"); k+=1)
            currentFile = StringFromList(k, indexList, ";")
            currentFile = RemoveEnding(currentFile, ".ibw")
            devar = StringFromList(0, currentFile, "_")
            strswitch(devar)
               case "Rl":
               case "Rxx":
               case "RxxDC":
                  LoadWave/H/P=binarypath currentFile+".ibw"
                  SetWaveLock 1, $currentFile
                  AppendToGraph $currentFile
                  ModifyGraph axRGB(left) = (65280,0,0)
                  if(strlen(leftlabel) == 0)
                     leftlabel = devar
                  else
                     leftlabel += ", " + devar
                  endif
                  indexList = RemoveFromList(currentFile + ".ibw", indexList, ";")
                  k -= 1
```

```
                        break
                    case "Rd":
                    case "Rdp":
                    case "Rdm":
                    case "RdDC":
                    case "Rxy":
                    case "RxyDC":
                        LoadWave/H/P=binarypath currentFile+".ibw"
                        SetWaveLock 1, $currentFile
                        AppendToGraph/R $currentFile
                        if(StringMatch(devar[0,1], "Rd"))
                            ModifyGraph rgb($currentFile) = (0, 0, 65280)
                        else
                            ModifyGraph rgb($currentFile) = (0, 0, 0)
                        endif
                        if(strlen(rightlabel) == 0)
                            rightlabel = devar
                        else
                            rightlabel += ", " + devar
                        endif
                        indexList = RemoveFromList(currentFile + ".ibw", indexList, ";")
                        k -= 1
                        break
                    endswitch
                endfor
                //update axis labels
                if(!stringmatch(leftlabel, ""))
                    Label left, leftlabel + " (" + devarunits + ")"
                endif
                if(!stringmatch(rightlabel, ""))
                    Label right, rightlabel + " (" + devarunits + ")"
                endif
                movewindow/I 0, 0, 2.95, 2.2
                DoWindow/B kwTopWin
                leftlabel = ""
                rightlabel = ""
                break
            default: //if wave is not a resistance, graph normally
                display $currentFile
                GraphStyle()
                Label left, devar + " (" + devarunits + ")"
                Label bottom, invar1 + " (" + invar1units + ")"
                windowleft = (mod(m+1, 3)*3)
                windowtop = (floor((m+1)/3)*3)
                windowright = (mod(m+1, 3)*3 + 2.95)
                windowbottom = (floor((m+1)/3)*3 + 2.2)
                movewindow/I windowleft, windowtop, windowright, windowbottom
                DoWindow/B kwTopWin
        endswitch
    endfor

    else
        printf "File name %s.ibw is not valid: not loading. (Loadibw)\r", currentFile
    endif
else
    if(n == LastIndex)
        LoadedIndex -= 1
    else
        printf "Unable to find waves with index %d.  Continuing to next index. (Loadibw)\r", n
    endif
endif
endfor
```

```
End

function wait(seconds)  //Much more accurate timer brought to you by JBM
   variable seconds
   seconds *= 1e6
   variable now=stopMStimer(-2)
   do
   while((stopMStimer(-2)-now) < seconds)
end

function nextwave([increment]) //return Index + 1
   variable increment //increment should only be used when called by do1d or do2d
   if(ParamIsDefault(increment))
      increment = 0
   endif
   wave/T ExperimentalDetails, ExperimentalDetailsName
   variable nextindex = NaN
   FindValue/TEXT="Index"/TXOP=4 ExperimentalDetailsName
   if(V_value > -1)
      nextindex = str2num(ExperimentalDetails[V_value][1]) + 1
      if(increment != 0) //if increment is nonzero, add 1 to Index and record new value
         ExperimentalDetails[V_value][1] = num2istr(nextindex)
         variable/G LoadedIndex
         //also update LoadedIndex to prevent loading graphs with loadibw in Run experiment
         LoadedIndex = nextindex
      endif
      return nextindex
   else
      Print "\"Index\" not found in ExperimentalDetails."
      Print "Unable to provide next index number. (NextWave)"
      return NaN
   endif
end

//graph styles for 1D and 2D waves
function GraphStyle()
   Legend/C/N=text0/F=0/A=LT/B=1
end

function GraphStyle2D()
   ModifyGraph margin(right)=72 //72 here is equal to 1.0 in the Modify Graph margins box
   ModifyGraph margin(left)=43 //(0.6)
// ColorScale/F=0/A=RC/X=-60/Y=5
end
```

C.8 Data Handling

An individual wave can be saved as an IGOR binary file (`.ibw`). Alternatively, one can use an IGOR text file (`.itx`), which has the advantage of also being able to contain a graph recreation macro. This section has a number of functions for saving and loading each type of file. The goal is to allow the user to easily and efficiently

transfer data between computers without having to work with the entire experiment file (which may be hundreds of MB and contain thousands of waves). The basic functions are `SaveGraph` and `LoadGraph`, which are used to save (as `.itx`) and load a single graph with all included waves. Other functions provide options to handle multiple files simultaneously.

```
#pragma rtGlobals=1     // Use modern global access method.

//loads all binary files with index in range defined in numberlist, does not graph
function LoadBinaryList(numberlist, [overwrite])
   string numberlist //; delimited list with ranges specified using -
   variable overwrite
   if(ParamIsDefault(overwrite))
      overwrite = 0  //does not overwrite file
   endif

   //fulllist is complete list of index numbers, individually ; delimited
   string fulllist = ";", currentitem = ""
   variable listsize = ItemsInList(numberlist, ";")
   variable n = 0, m = 0, startnum = 0, endnum = 0
   //parse numberlist to expand ranges given using -
   for(n=0; n<listsize; n+=1)
      currentitem = StringFromList(n, numberlist, ";")
      if(GrepString(currentitem, "^\d+$")) //if list item is a single number append to fulllist
         fulllist += currentitem + ";"
//if list item is a range append full range, inclusive, to fullist
      elseif(GrepString(currentitem, "^\d+\-\d+$"))
         sscanf currentitem, "%d-%d", startnum, endnum
         for(m=startnum; m<endnum+1; m+=1)
            fulllist += num2istr(m) + ";"
         endfor
      else //any other format is wrong
         printf "List item %s is not valid.  ", currentitem
         printf "Use single numbers or ranges (i.e. \"4-15\") only. (LoadBinaryList)\r"
      endif
   endfor

   string ibwList = IndexedFile(binarypath, -1, ".ibw") //get list of all .ibw files
   listsize = ItemsInList(ibwList, ";")
   variable match1 = 0, match2 = 0
   for(n=0; n<listsize; n+=1)
      currentitem = StringFromList(n, ibwlist, ";")
      currentitem = RemoveEnding(currentitem, ".ibw")
      //check if index is included in fulllist
      match1 = stringmatch(fulllist, "*;"+StringFromList(2, currentitem, "_")+";*")
      match2 = stringmatch(fulllist, "*;"+StringFromList(3, currentitem, "_")+";*")
      if(match1 || match2)
         currentitem += ".ibw"
         if(overwrite == 0)
            LoadWave/H/P=binarypath currentitem //load wave, no overwrite
         else
            LoadWave/H/O/P=binarypath currentitem //load wave, overwrite existing wave
         endif
      endif
   endfor
end
```

```
//saves top graph and waves as Igor text file
function SaveGraph(gname, [overwrite, nomacro, windowname])
   //name to save as, name of target window (optional, defaults to top window)
   string gname, windowname
   variable overwrite, nomacro
   if(ParamIsDefault(overwrite))
      overwrite = 0  //does not overwrite file
   endif
   if(ParamIsDefault(nomacro))
      nomacro = 0 //0: creates recreation macro, 1: do not create recreation macro
      //recreation macro functionality is currently disabled
   endif
   //recommended to use nomacro = 1 when SaveGraph called by another function
   //must use nomacro = 1 when called in loop or risk incorrect macros (due to Execute/P)
   variable topwin = 0
   if(ParamIsDefault(windowname))
      topwin = 1 //0: use windowname to choose target window, 1: target top window
   endif

   //check that top window is indeed a graph
   if(topwin == 1)
      if(WinType("") != 1)
         Print "Top window not a graph.  Aborting save. (SaveGraph)"
         return -1
      endif
   else
      if(WinType(windowname) != 1)
         Print "Target window " + windowname + " not a graph.  Aborting save. (SaveGraph)"
         return -1
      endif
   endif

   string commandstr, graphstring = "", singleline = ""
   string wlist //list of waves in graph
   string savename = gname + ".itx" //.itx is for Igor text files
   variable gslines = 0 //number of lines in graph recreation macro
   make/O/T/n=0 goutwave //graph recreation commands to be appended at end of file
   variable n = 0

   //rename window according to gname
   if(topwin == 1)
      commandstr = "DoWindow/C " + gname
   else
      commandstr = "DoWindow/C/W=" + windowname + " " + gname
   endif
   Execute/Q commandstr

   commandstr = "GetWindow " + gname + ", wavelist" //writes list of waves in graph to W_WaveList
   Execute/Q commandstr
   wlist = TWave2Str(W_WaveList) //generate wlist
   //get window recreation commands and put in correct format
   if(topwin == 1)
      graphstring = WinRecreation("",0)
   else
      graphstring = WinRecreation(gname,0)
   endif
   gslines = ItemsInList(graphstring,"\r")
   for(n=2;n<gslines-1;n+=1)
      InsertPoints n-1, 1, goutwave //add another point to goutwave
      //remove "\t" characters
      singleline = "X " + StringFromList(1,StringFromList(n,graphstring,"\r"),"\t")
      singleline = ReplaceString("\\\\", singleline, "\\")
```

```
      singleline = ReplaceString("\\r", singleline, "\r")
      goutwave[n-2] = singleline
      //.itx commands start with "X "
   endfor
   InsertPoints n,1,goutwave
   goutwave[inf] = "X DoWindow/C " + gname
// print wlist
   //save waves as text file and append graph recreation macro
   if(overwrite == 0)
      sprintf commandstr "Save/T/B/P=graphpath \"%s\" as \"%s\";", wlist, savename
   else
      sprintf commandstr "Save/O/T/B/P=graphpath \"%s\" as \"%s\";", wlist, savename
   endif
   Execute/Q commandstr
   variable numlines = DimSize(goutwave, 0)
   for(n=0; n<numlines; n+=1)
      Duplicate/O/R=[n,n] goutwave, SaveGraphWriteWave
      sprintf commandstr "Save/A=2/G/P=graphpath SaveGraphWriteWave as \"%s\"", savename
      Execute/Q commandstr
   endfor

   //save graph recreation macro
   //uncomment these lines to restore this functionality
   //also make sure to create GraphMacros.ipf manually or in Initializations or something
// if(nomacro==0)
//    make/O/T gmacro
//    string grepsearch = "Window " + gname + "()" //string to use to search for graph macro
//    Grep/O/P=graphpath/E=grepsearch "GraphMacros.ipf" as gmacro //search for graph macro
//    if(strlen(TWave2Str(gmacro))==0) //graph macro not already saved
//       make/O/T/n=3 macrotext //recreaction commands
//       macrotext[0] = "Window " + gname + "() : Graph"
//       macrotext[1] = "LoadGraph(\"" + gname + "\")" //recreate graph with LoadGraph
//       macrotext[2] = "EndMacro"
         //have to close procedure file before writing to it
//       commandstr = "CloseProc/Name=\"GraphMacros.ipf\""
         //WARNING: Execute/P sends commands to a queue which executes when no procedures are running
//       Execute/P/Q commandstr
         //write recreation macro to file
//       commandstr = "Save/A/G/P=graphpath macrotext as \"GraphMacros.ipf\""
//       Execute/P/Q commandstr
//       commandstr = "OpenProc/V=0/P=graphpath \"GraphMacros.ipf\"" //reopen file
//       Execute/P/Q commandstr
//    endif
// endif
end

function LoadGraph(gname, [overwrite]) //loads waves and graph from saved file
   string gname //file (and graph) name
   variable overwrite
   string commandstr
   sprintf commandstr "DoWindow %s", gname //sets V_flag = 1 or 2 if graph exists
   Execute commandstr
   NVAR windowexists = V_flag
   if(windowexists > 0)
      Print "Graph " + gname + " already exists.  Not loading saved copy. (LoadGraph)"
      return -1
   endif
   if(ParamisDefault(overwrite))
      overwrite = 0 //does not automatically overwrite waves if they already exist
   endif

   string filestring = gname + ".itx" //.itx = Igor text file
```

```
   if(overwrite == 0)
      sprintf commandstr "LoadWave/Q/T/P=graphpath \"%s\"", filestring
   else
      sprintf commandstr "LoadWave/Q/O/T/P=graphpath \"%s\"", filestring
   endif
   Execute/Z/Q commandstr //loads waves and executes graph recreation commands in file
end

function Archive([aname]) //saves all graphs with names of form "Graph#" as indexed .itx files
   string aname //name of file to record list of open graphs
   variable recordopen = 1 //archive record of open graphs
   if(ParamIsDefault(aname))
      recordopen = 0 //don't record open graphs
   endif

   string killlist = DataWaveList() //list of all data waves (with name matching proper format)
   variable killlength = ItemsInList(killlist, ";")
   variable n = 0, m = 0
   string wavetokill = ""
   string savedfiles = ""
   variable numsavedfiles = 0, archiveindex = 0, indexlength = 0
   string testname = "", archivename = ""
   string graphlist = WinList("Graph*", ";", "WIN:1") //list of graphs with names of form "Graph*"
   variable numgraphs = ItemsInList(graphlist, ";")
   string gnametocheck = ""
   string commandstr = ""

   //kill all data waves not in use
   for(n=0; n<killlength; n+=1)
      wavetokill = StringFromList(n, killlist, ";")
      KillWaves/Z $wavetokill //does not kill waves which are in graphs, tables, etc.
   endfor

   savedfiles = IndexedFile(graphpath, -1, ".itx") //list of .itx files in folder given by graphpath
   numsavedfiles = ItemsInList(savedfiles, ";")
   n = numsavedfiles - 1
   //look for saved file with greatest index,
   //assumes files are returned by IndexedFile in alphabetical order
   do
      testname = StringFromList(n, savedfiles, ";")
      if(GrepString(testname, "Archive\d{4}"))
         //set testname equal to 4 digit number at end of file name
         sscanf testname, "Archive%d", archiveindex
         break
      endif
      n -= 1
   while(n > -1)

   //saves graphs with an increasing index
   for(n=0; n<numgraphs; n+=1)
      gnametocheck = StringFromList(n, graphlist, ";")
      //only saves graphs with names of form "Graph#", where # is a nonnegative integer
      if(GrepString(gnametocheck, "Graph\d{1,}"))
         archiveindex += 1
         archivename = "Archive"
         indexlength = strlen(num2str(archiveindex))
         if(indexlength<4) //index is always a 4 digit number, with leading 0s if necessary
            for(m=indexlength; m<4; m+=1)
               archivename += "0"
            endfor
         endif
         archivename += num2str(archiveindex)
         SaveGraph(archivename, nomacro = 1, windowname = gnametocheck) //saves graph
```

```
      endif
   endfor

   //saves list of open graphs using aname as identifier
   if(recordopen == 1)
      graphlist = WinList("*", ";", "WIN:1") //list of all open graphs
      numgraphs = ItemsInList(graphlist, ";")
      make/O/T/n=(numgraphs) glistout
      for(n=0; n<numgraphs; n+=1) //; delimited list
         glistout[n] = StringFromList(n, graphlist, ";")
      endfor
      aname += ".atx" //arbitrary extension: "Archive Text file", same format as .itx
      sprintf commandstr "Save/O/T/P=graphpath glistout as \"%s\"", aname
      Execute/Q commandstr
   endif
end

//save graphs with names of the form Graph#, for the specified numbers
function ArchiveList(numberlist, aname)
   string numberlist, aname //numberlist: ; delimited list with ranges specified using -
      //aname: name for .atx file to record names of saved graphs (for extraction)

   //fulllist is complete list of graph numbers, individually ; delimited
   string fulllist = "", currentitem = ""
   variable listsize = ItemsInList(numberlist, ";")
   variable n = 0, m = 0, startnum = 0, endnum = 0
   //parse numberlist to expand ranges given using -
   for(n=0; n<listsize; n+=1)
      currentitem = StringFromList(n, numberlist, ";")
      if(GrepString(currentitem, "^\d+$")) //if list item is a single number append to fulllist
         fulllist += currentitem + ";"
      //if list item is a range append full range, inclusive, to fullist
      elseif(GrepString(currentitem, "^\d+\-\d+$"))
         sscanf currentitem, "%d-%d", startnum, endnum
         for(m=startnum; m<endnum+1; m+=1)
            fulllist += num2istr(m) + ";"
         endfor
      else //any other format is wrong
         printf "List item %s is not valid.  ", currentitem
         printf "Use single numbers or ranges (i.e. \"4-15\") only. (ArchiveList)\r"
      endif
   endfor

   //list of .itx files in folder given by graphpath
   string savedfiles = IndexedFile(graphpath, -1, ".itx")
   variable numsavedfiles = ItemsInList(savedfiles, ";")
   string testname = ""
   variable archiveindex = 0
   //look for saved file with greatest index,
   //assumes files are returned by IndexedFile in alphabetical order
   n = numsavedfiles - 1
   do
      testname = StringFromList(n, savedfiles, ";")
      if(GrepString(testname, "Archive\d{4}"))
         //set archiveindex equal to 4 digit number at end of file name
         sscanf testname, "Archive%d", archiveindex
         break
      endif
      n -= 1
   while(n > -1)

   //saves graphs with an increasing index
   string gnametocheck = "", archivename = ""
```

```
   variable indexlength = 0
   variable numgraphs = ItemsInList(fulllist, ";")
   make/O/T/n=0 glistout //wave to save with list of saved graphs
   for(n=0; n<numgraphs; n+=1)
      gnametocheck = "Graph" + StringFromList(n, fulllist, ";")
      DoWindow $gnametocheck //returns V_flag = 0 if window does not exist
      if(V_flag != 0) //check that numbered graph actually exists
         archiveindex += 1
         archivename = "Archive"
         indexlength = strlen(num2str(archiveindex))
         if(indexlength<4) //index is always a 4 digit number, with leading 0s if necessary
            for(m=indexlength; m<4; m+=1)
               archivename += "0"
            endfor
         endif
         archivename += num2str(archiveindex)
         SaveGraph(archivename, nomacro = 1, windowname = gnametocheck) //saves graph
         InsertPoints Inf, 1, glistout //insert point at end of glistout
         glistout[Inf] = archivename //record name of saved graph
      else
         printf "Cannot find %s.  Not saving. (ArchiveList).\r", gnametocheck
      endif
   endfor

   //saves list of open graphs using aname as identifier
   aname += ".atx" //arbitrary extension: "Archive Text file", same format as .itx
   string commandstr = ""
   sprintf commandstr "Save/O/T/P=graphpath glistout as \"%s\"", aname
   Execute/Q commandstr
end

function ExtractNew([overwrite]) //loads graphs that do not match names of graphs already open
   variable overwrite
   if(ParamIsDefault(overwrite))
      overwrite = 0 //do not overwrite waves of the same name
   endif

   string graphlist = WinList("*", ";", "WIN:1") //list of all graphs
   variable numgraphs = ItemsInList(graphlist, ";")
   string filelist = IndexedFile(graphpath, -1, ".itx") //list of all .itx files
   variable numfiles = ItemsInList(filelist, ";")
   variable n = 0, m = 0
   string currentfile = ""
   variable namelength = 0
   variable loadflag = 0

   //load graphs and waves
   for(n=0; n<numfiles; n+=1)
      loadflag = 1
      currentfile = StringFromList(n, filelist, ";")
      namelength = strlen(currentfile)
      currentfile = currentfile[0,namelength-5] //remove .itx extension
      for(m=0; m<numgraphs; m+=1)
         if(stringmatch(currentfile, StringFromList(m, graphlist, ";")))
            loadflag = 0
         endif
      endfor
      if(loadflag == 1)
         LoadGraph(currentfile, overwrite = overwrite)
         if(GrepString(currentfile, "Archive\d{1,}"))
            DoWindow/B kwTopWin
         endif
      endif
```

```
   endfor
end

function ExtractNamed(namelist, [overwrite]) //loads graphs from a ; delimited list
   string namelist //; delimited list of archive files to extract
   variable overwrite
   if(ParamIsDefault(overwrite))
      overwrite = 0 //do not overwrite waves of same name
   endif

   variable listlength = ItemsInList(namelist, ";")
   variable n = 0
   string currentname = "", notloaded = ""

   //load graphs and waves
   for(n=0; n<listlength; n+=1)
      currentname = StringFromList(n, namelist, ";")
      GetFileFolderInfo/Q/Z/P=graphpath (currentname + ".itx") //sets V_flag = 0 if named file exists
      if(V_flag == 0)
         LoadGraph(currentname, overwrite = overwrite)
      else
         notloaded += currentname + ";" //if file not found add to notloaded list
      endif
   endfor

   //print list of files not found
   if(strlen(notloaded) > 0)
      Print "The following could not be found: " + notloaded + ". (ExtractNamed)"
   endif
end

function ExtractRange(startnum, endnum, [overwrite]) //loads graghs with index in range
   variable startnum, endnum //start and end of range, inclusive
   variable overwrite
   if(endnum<startnum)
      print "Second integer must be greater than first integer. (ExtractRange)"
      return -1
   endif
   if(ParamIsDefault(overwrite))
      overwrite = 0 //do not overwrite waves of same name
   endif

   variable n = 0, m = 0
   string filename = "", notloaded = ""
   variable numlength = 0

   //load graphs and waves
   for(n=startnum; n<endnum+1; n+=1)
      filename = "Archive"
      numlength = strlen(num2str(n))
      if(numlength<4)
         for(m=numlength; m<4; m+=1)
            filename += "0"
         endfor
      endif
      filename += num2str(n)
      GetFileFolderInfo/Q/Z/P=graphpath (filename + ".itx") //sets V_flag = if named file exits
      if(V_flag == 0)
         LoadGraph(filename, overwrite = overwrite)
         DoWindow/B kwTopWin
      else
         notloaded += filename + ";" //if file not found add to notloaded list
      endif
```

```
   endfor

   //print list of files not found
   if(strlen(notloaded) > 0)
      Print "The following could not be found: " + notloaded + ". (ExtractRange)"
   endif
end

function ExtractAll([overwrite]) //loads all graphs in folder given by graphpath
   variable overwrite
   if(ParamIsDefault(overwrite))
      overwrite = 0 //do not overwrite waves of same name
   endif

   string filelist = IndexedFile(graphpath, -1, ".itx") //list of all .itx files
   variable numfiles = ItemsInList(filelist, ";")
   variable n = 0
   string currentfile = ""
   variable namelength = 0

   //load graphs and waves
   for(n=0; n<numfiles; n+=1)
      currentfile = StringFromList(n, filelist, ";")
      namelength = strlen(currentfile)
      currentfile = currentfile[0,namelength-5] //remove .itx extension
      LoadGraph(currentfile, overwrite = overwrite)
      if(GrepString(currentfile, "Archive\d{1,}"))
         DoWindow/B kwTopWin
      endif
   endfor
end

function ExtractAllNamed([overwrite]) //load all graphs with names not of form "Archive####"
   variable overwrite
   if(ParamIsDefault(overwrite))
      overwrite = 0 //do not overwrite waves of same name
   endif

   string filelist = IndexedFile(graphpath, -1, ".itx") //list of all .itx files
   variable numfiles = ItemsInList(filelist, ";")
   variable n = 0
   string currentfile = ""
   variable namelength = 0

   //load graphs and waves
   for(n=0; n<numfiles; n+=1)
      currentfile = StringFromList(n, filelist, ";")
      if(!GrepString(currentfile, "Archive\d{1,}")) //don't load indexed files
         namelength = strlen(currentfile)
         currentfile = currentfile[0,namelength-5]
         LoadGraph(currentfile, overwrite = overwrite)
      endif
   endfor
end

function ExtractArchive(aname, [overwrite]) //loads all graphs listed in aname.atx
   string aname
   variable overwrite
   if(ParamIsDefault(overwrite))
      overwrite = 0 //do not overwrite waves of same name
   endif

   string filelist = ""
```

```
   variable numfiles = 0
   variable n = 0
   string currentfile = "", notloaded = ""

   aname += ".atx"
   LoadWave/O/T/P=graphpath aname //creates glistout wave which lists graphs to load
   filelist = TWave2Str(glistout, separator = ";")
   numfiles = ItemsInList(filelist, ";")

   //load graphs and waves
   for(n=0; n<numfiles; n+=1)
      currentfile = StringFromList(n, filelist, ";")
      GetFileFolderInfo/Q/Z/P=graphpath (currentfile + ".itx") //sets V_Flag = 0 if file exists
      if(V_Flag == 0)
         LoadGraph(currentfile, overwrite = overwrite)
         if(GrepString(currentfile, "Archive\d{1,}"))
            DoWindow/B kwTopWin
         endif
      else
         notloaded += currentfile + ";" //if file not found add to notloaded list
      endif
   endfor

   //print list of files not found
   if(strlen(notloaded) > 0)
      Print "The following could not be found: " + notloaded
   endif
end

//returns ; delimited list of all waves with name format matching collected data
function/S DataWaveList()
   string allwaves = WaveList("*_*_*", ";","") //list of all waves with general format
   variable listlength = ItemsInList(allwaves, ";")
   variable n = 0
   string matchexp = "", currentstr = "", outstring = ""
   //lists of independent and dependent variables; | delimited for use as regex alternative branches
   SVAR InVarExp, DeVarExp

   //update InVarExp and DeVarExp to match current independent and dependent variable lists
   UpdateVariableLists()
   matchexp = "(" + DeVarExp + ")(_(" + InVarExp + ")){1,2}_\d{1,}" //regex expression to match names

   //check each wave if match data wave format
   outstring = GrepList(allwaves, matchexp)
   return outstring
end

//updates InVarExp and DeVarExp as | delimited lists of independent and dependent variables
function UpdateVariableLists()
   wave/T InVarWave, DeVarWave //waves of independent and dependent variables and corresponding units
   variable numinvar = DimSize(InVarWave, 0)
   variable numdevar = DimSize(DeVarWave, 0)
   variable n = 0
   string/G InVarList = "", DeVarList = "-;", InVarExp = "", DeVarExp = ""

   //update lists
   for(n=0; n<numinvar; n+=1)
      InVarList += InVarWave[n][0] + ";"
      InVarExp += InVarWave[n][0] + "(\d{1,}[[:alpha:]]{0,1}){0,1}|"
   endfor
   for(n=0; n<numdevar; n+=1)
      DeVarList += DeVarWave[n][0] + ";"
      DeVarExp += DeVarWave[n][0] + "|"
```

```
   endfor
   InVarExp=InVarExp[0,strlen(InVarExp)-2] //remove ending | to get regex to work right
      //use RemoveEnding instead?
   DeVarExp=DeVarExp[0,strlen(DeVarExp)-2]
end

function/S TWave2Str(wname, [separator]) //returns first column of text wave wname as list
   wave/T wname
   string separator
   if(ParamIsDefault(separator))
      separator=";"
   endif

   if(WaveType(wname)!=0)
      print "Error: " + NameOfWave(wname) + " not a text wave.  Returning \"\". (TWave2Str)"
      return ""
   endif

   string sname = ""
   variable n=0
   string currentcell = ""
   for(n=0; n<DimSize(wname,0); n+=1)
      sname += wname[n] + separator
   endfor
   return sname
end
```

C.9 Data Searching

These functions provide the ability to search through recorded experimental parameters to find waves that were taken under the desired conditions. The majority of the functionality is provided by `SearchPanel`, which provides a user interface to control the search.

```
#pragma rtGlobals=1     // Use modern global access method.

//-------------------------------------------------------------------
//  A series of functions to get information from DetailsSave file
//-------------------------------------------------------------------

function notediff(note1, note2)
   string note1, note2

   string str1 = "", str2 = ""

   Notebook $note1 selection = {startofFile, endofFile}
   GetSelection notebook, $note1, 2
   str1 = S_Selection
   Notebook $note2 selection = {startofFile, endofFile}
   GetSelection notebook, $note2, 2
   str2 = S_Selection
```

```
   variable len1 = strlen(str1), len2 = strlen(str2)

   printf "Length 1: %d, Length 2: %d\r", len1, len2

   variable n = 0
   for(n=0; n<min(len1, len2); n+=1)
      if(cmpstr(str1[n], str2[n]) != 0)
         printf "Strings diverge at index %d.  ", n
         printf "String 1: \"%s\", String 2: \"%s\"\r", str1[n-5, n+5], str2[n-5, n+5]
         return 0
      endif
   endfor
   print "No differences found."
end

//prints index numbers and dependent variable types of waves taken on given date
Function DataonDay(Sdate)
   string Sdate //Sdate in form of "Mon Day, Year"

   WAVE/T ExperimentalDetails
   Duplicate/O/R=[][0] ExperimentalDetails ExperimentalDetailsName
   variable numcols = DimSize(ExperimentalDetails, 1)

   FindValue/TEXT=("Date")/TXOP=4 ExperimentalDetailsName //find row with date data
   if(V_value > -1)
      variable daterow = V_value
      variable indexrow = -1, dependentrow = -1
      FindValue/TEXT=("Index")/TXOP=4 ExperimentalDetailsName //find row with index data
      if(V_value > -1)
         indexrow = V_value
      else
         Print "\"Index\" not found in ExperimentalDetails.  Not printing index. (DataonDay)"
      endif
      FindValue/TEXT=("Dependent")/TXOP=4 ExperimentalDetailsName //find row with dependent variables
      if(V_value > -1)
         dependentrow = V_value
      else
         Print "\"Dependent\" not found in ExperimentalDetails."
         Print "Not printing dependent variables. (DataonDay)"
      endif

      variable n = 3
      string detailsdate, output = ""
      //search columns for matching dates
      for(n=3; n<numcols; n+=1)
         detailsdate =  ExperimentalDetails[daterow][n]
         if(StringMatch(Sdate, detailsdate[5, 16]))
            output = ""
            if(indexrow != -1)
               //include index to be printed
               output += "Index: " + ExperimentalDetails[indexrow][n] + "; "
            endif
            if(dependentrow != -1)
               //include dependent variables to be printed
               output += "Dependent: " + ExperimentalDetails[dependentrow][n]
            endif
            Print output
         endif
      endfor
   else
      Print "\"Date\" not found in ExperimentalDetails.  Unable to get trace info. (DataonDay)"
   endif
end
```

```
function SearchPanel([numsearch]) : Panel //panel to search ExperimentalDetails wave
   variable numsearch //number of non-dependent variable search options
   if(ParamIsDefault(numsearch))
      numsearch = 3
   endif

   variable/G SearchPanelNum = numsearch
   //define most common search options
   string/G SearchPanelTypes = "-;Date;Time;Index;BStart;BStop;"
   SearchPanelTypes += "IdcStart;IdcStop;TimeRate;VgStart;VgStop"
   NewPanel /W=(3,0,403,125 + numsearch*25) as "SearchPanel"
   variable n = 0
   string commandstr = "", commandstr1 = "", commandstr2 = "", commandstr3 = "", commandstr4 = ""
   string commandstr5 = "", commandstr6 = "", commandstr7 = "", commandstr8 = ""
   string commandstr1b = "", commandstr4b = ""
   //create items for each search option, number given by numsearch
   for(n = 0; n<numsearch; n+=1)
      sprintf commandstr "string/G V_from%d = \"\", V_to%d = \"the same\"", n, n
      sprintf commandstr1 "string/G SearchPanelTypes%d = SearchPanelTypes, ", n
      sprintf commandstr1b "SearchPanelInput%d = \"-\"", n
      commandstr1 += commandstr1b
      sprintf commandstr2 "CheckBox inputall%d,pos={3,%d+3},", n, n*25
      commandstr2 += "size={14,14},proc=InputCheck,title=\" \""
      sprintf commandstr3 "PopupMenu inputtype%d,pos={93,%d},", n, n*25
      commandstr3 += "size={61,31},proc=SetInputType, title = \" \""
      sprintf commandstr4 "PopupMenu inputtype%d,mode=1,bodyWidth=130,"
      sprintf commandstr4b "popvalue=\"Search Type\",value= #\"SearchPanelTypes%d\"", n, n
      commandstr4 += commandstr4b
      sprintf commandstr5 "SetVariable inputfrom%d,pos={220,1+%d},size={61,31},", n, n*25
      commandstr5 += "proc=SetInputValue,title=\"From\""
      sprintf commandstr6 "SetVariable inputfrom%d,bodyWidth=80,disable=1,value=V_from%d", n, n
      sprintf commandstr7 "SetVariable inputto%d,pos={330,1+%d},size={61,31},", n, n*25
      commandstr7 += "proc=SetInputValue,title=\"To\""
      sprintf commandstr8 "SetVariable inputto%d,bodyWidth=80,disable=1,value=V_to%d", n, n
      Execute/Q commandstr
      Execute/Q commandstr1
      Execute/Q commandstr2
      Execute/Q commandstr3
      Execute/Q commandstr4
      Execute/Q commandstr5
      Execute/Q commandstr6
      Execute/Q commandstr7
      Execute/Q commandstr8
   endfor

   DrawLine 0,n*25,1200,n*25

   //create checkboxes for searching by dependent variable type
   variable/G SearchPanelBmeas = 0, SearchPanelRd = 0, SearchPanelRxx = 0, SearchPanelRxy = 0
   variable/G SearchPanelOpen
   string/G V_open
   CheckBox BmeasCheck,pos={3,3+(n+0)*25},size={73,14},variable=SearchPanelBmeas,title="Bmeas"
   CheckBox RdCheck,pos={3,3+(n+1)*25},size={73,14},variable=SearchPanelRd,title="Rd"
   CheckBox RxxCheck,pos={3,3+(n+2)*25},size={73,14},variable=SearchPanelRxx,title="Rxx"
   CheckBox RxyCheck,pos={3,3+(n+3)*25},size={73,14},variable=SearchPanelRxy,title="Rxy"
   CheckBox OpenCheck,pos={3,3+(n+4)*25},size={14,14},variable=SearchPanelOpen,title=""
   SetVariable OpenInput,pos={40,1+(n+4)*25},size={61,31},bodywidth=80,value=V_open,title=" "

   //create inputs for searching by independent variable type
   string/G SearchPanelInVar1 = "", SearchPanelInVar2 = ""
   variable/G SearchPanelInVarSym = 0
   UpdateVariableLists()
```

```
   SVAR InVarList
   string InVarMenu = "\"-;" + InVarList + ";\\w{1,}" + "\""
   PopupMenu invar1,pos={113,(n+1)*25},size={41,31},noproc, title = ""
   PopupMenu invar1,mode=2,bodyWidth=90,popvalue = "InVar 1",value= #InVarMenu
   PopupMenu invar2,pos={113,(n+2)*25},size={41,31},noproc, title = ""
   PopupMenu invar2,mode=1,bodyWidth=90,popvalue="InVar 2",value= #InVarMenu
   CheckBox ivsymchk,pos={63,3+(n+3)*25},size={73,14},variable=SearchPanelInVarSym,title="Symmetric"

   //button to start search
   Button dosearch, pos={93,n*25+3},size={60,17},proc=GetTable,title="Click Me"
   Button dosearch, fstyle=1,fColor=(100,50000,10000)
   DrawLine 155,n*25,155,n*25+150

   //checkboxes to define which types of output are included
   variable/G V_all = 0, V_numpoints = 0, V_ivsource = 0
   variable/G V_scalefactors = 0, V_lipa = 0, V_temp = 0
   CheckBox AllChk,pos={158,3+(n+0)*25},size={73,14},title="Everything",proc=RowsCheck,variable=V_all
   CheckBox NumPointsCheck,pos={158,3+(n+1)*25},size={73,14},title="Num Points",variable=V_numpoints
   CheckBox IVSourceCheck,pos={158,3+(n+2)*25},size={73,14},title="I/V Source",variable=V_ivsource
   CheckBox SFsCheck,pos={158,3+(n+3)*25},size={73,14},title="Scale Factors",variable=V_scalefactors
   CheckBox LIPACheck,pos={158,3+(n+4)*25},size={73,14},title="LI/PA",variable=V_lipa
   CheckBox TempCheck,pos={270,3+(n+4)*25},size={73,14},title="Temp",variable=V_temp

   //ouput types relating to independent variables
   variable/G V_binfo = 0, V_idcinfo = 0, V_timeinfo = 0, V_vginfo = 0
   string/G V_vgnums
   CheckBox BInfoCheck,pos={270,3+(n+0)*25},size={73,14},title="B Info",variable=V_binfo
   CheckBox IdcInfoCheck,pos={270,3+(n+1)*25},size={73,14},title="I/Vdc Info",variable=V_idcinfo
   CheckBox TimeInfoCheck,pos={270,3+(n+2)*25},size={73,14},title="Time Info",variable=V_timeinfo
   CheckBox VgCheck,pos={270,3+(n+3)*25},size={73,14},title="Vg",variable=V_vginfo
   SetVariable VgInput,pos={306,1+(n+3)*25},size={50,31},bodywidth=50,value=V_vgnums,title=" "

   variable/G V_binfoall = 0, V_idcinfoall = 0, V_timeall = 0, V_vginfoall = 0
   CheckBox BChkAll,pos={360,3+(n+0)*25},size={73,14},title=" ",proc=RowsCheck,variable=V_binfoall
   CheckBox IdcChkAll,pos={360,3+(n+1)*25},size={73,14},title=" ",proc=RowsCheck,variable=V_idcinfoall
   CheckBox TimeChkAll,pos={360,3+(n+2)*25},size={73,14},title=" ",proc=RowsCheck,variable=V_timeall
   CheckBox VgCheckAll,pos={360,3+(n+3)*25},size={73,14},title=" ",proc=RowsCheck,variable=V_vginfoall
end

//updates SearchPanel in response to certain output checkboxes
function RowsCheck(ctrlName,checked) : CheckBoxControl
   string ctrlName
   variable checked

   string commandstr = ""
   string menunum = ctrlName[8]

   //disable other output checkboxes when "Everything" is checked
   if(StringMatch(ctrlName, "AllChk"))
      if(checked == 1)
         CheckBox NumPointsCheck,disable=2
         CheckBox IVSourceCheck,disable=2
         CheckBox SFsCheck,disable=2
         CheckBox LIPACheck,disable=2
         CheckBox TempCheck,disable=2
         CheckBox BInfoCheck,disable=2
         CheckBox IdcInfoCheck,disable=2
         CheckBox TimeInfoCheck,disable=2
         CheckBox VgCheck,disable=2
         SetVariable VgInput,disable=2
         CheckBox BChkAll,disable=2
         CheckBox IdcChkAll,disable=2
         CheckBox TimeChkAll,disable=2
```

```
         CheckBox VgCheckAll,disable=2
      else
         CheckBox NumPointsCheck,disable=0
         CheckBox IVSourceCheck,disable=0
         CheckBox SFsCheck,disable=0
         CheckBox LIPACheck,disable=0
         CheckBox TempCheck,disable=0
         CheckBox BInfoCheck,disable=0
         CheckBox IdcInfoCheck,disable=0
         CheckBox TimeInfoCheck,disable=0
         CheckBox VgCheck,disable=0
         SetVariable VgInput,disable=0
         CheckBox BChkAll,disable=0
         CheckBox IdcChkAll,disable=0
         CheckBox TimeChkAll,disable=0
         CheckBox VgCheckAll,disable=0
      endif
   //disable appropriate checkboxes when independent variable "All" boxes checked
   elseif(StringMatch(ctrlName, "BChkAll"))
      if(checked == 1)
         CheckBox BInfoCheck,disable=2
      else
         CheckBox BInfoCheck,disable=0
      endif
   elseif(StringMatch(ctrlName, "IdcChkAll"))
      if(checked == 1)
         CheckBox IdcInfoCheck,disable=2
      else
         CheckBox IdcInfoCheck,disable=0
      endif
   elseif(StringMatch(ctrlName, "TimeChkAll"))
      if(checked == 1)
         CheckBox TimeInfoCheck,disable=2
      else
         CheckBox TimeInfoCheck,disable=0
      endif
   elseif(StringMatch(ctrlName, "VgCheckAll"))
      if(checked == 1)
         CheckBox VgCheck,disable=2
      else
         CheckBox VgCheck,disable=0
      endif
   endif
end

//controls number of items displayed in input dropdown menus
function InputCheck(ctrlName,checked) : CheckBoxControl
   string ctrlName
   variable checked

   string commandstr = ""
   string menunum = ctrlName[8]

   if(checked == 1)
      //display all items in ExperimentDetails
      sprintf commandstr, "SearchPanelTypes%s = \"-;\" + ", menunum
      commandstr += "TWave2Str(ExperimentalDetails, separator = \";\")"
      Execute/Q commandstr
   else
      //display only common choices defined when creating SearchPanel
      sprintf commandstr, "SearchPanelTypes%s = SearchPanelTypes", menunum
      execute/Q commandstr
   endif
```

```
end

//updates the "To" text box if no user-defined text is present
Function SetInputValue(ctrlName,varNum,varStr,varName) : SetVariableControl
   String ctrlName
   Variable varNum   // value of variable as number
   String varStr     // value of variable as string
   String varName // name of variable

   string commandstr = ""

   if(StringMatch(ctrlName[5,6], "to"))
      if(StringMatch(varStr, ""))
      //update back to default, which is recognized while searching
      sprintf commandstr "%s = \"the same\"", varName
      Execute/Q commandstr
      endif
   endif
End

//controls visibility of "From" and "To" text boxes based on if a search parameter is chosen
Function SetInputType(ctrlName,popNum,popStr) : PopupMenuControl
   String ctrlName
   Variable popNum   // which item is currently selected (1-based)
   String popStr     // contents of current popup item as string

   string commandstr = "", commandstr2 = ""
   string menunum = ctrlName[9]
   sprintf commandstr "SearchPanelInput%s = \"%s\"", menunum, popStr
   Execute/Q commandstr

   if(!StringMatch(popStr, "-"))
      //make corresponding "From" and "To" text boxes invisible
      sprintf commandstr, "SetVariable inputfrom%s, disable = 0; ", menunum
      commandstr2 = "SetVariable inputto%s, disable = 0", menunum
      commandstr += commandstr2
      Execute/Q commandstr
   else
      //make corresponding "From" and "To" text boxes visible
      sprintf commandstr, "SetVariable inputfrom%s, disable = 1; ", menunum
      commandstr2 = "SetVariable inputto%s, disable = 1", menunum
      commandstr += commandstr2
      Execute/Q commandstr
   endif
end

//search ExperimentalDetails and output results in ExperimentalDetailsSearch
Function GetTable (ctrlName) : ButtonControl
   String ctrlName

   NVAR SearchPanelNum
   NVAR SearchPanelBmeas, SearchPanelRd, SearchPanelRxx, SearchPanelRxy, SearchPanelOpen
   SVAR V_open
   NVAR V_all, V_numpoints, V_ivsource, V_scalefactors, V_lipa, V_temp
   NVAR V_binfo, V_idcinfo, V_timeinfo, V_vginfo
   SVAR V_vgnums
   NVAR V_binfoall, V_idcinfoall, V_timeall, V_vginfoall

   WAVE/T ExperimentalDetails
   Duplicate/T/O/R=[][0] ExperimentalDetails ExperimentalDetailsName
   Duplicate/T/O ExperimentalDetails ExperimentalDetailsSearch
   DeletePoints/M=1 1, 1, ExperimentalDetailsSearch //remove 'current status' column
   //remove duplicate 'name' column at end
```

```
DeletePoints/M=1 (DimSize(ExperimentalDetailsSearch,1) - 1), 1, ExperimentalDetailsSearch

variable n = 0, m = 2, k = 0
string inputvar = "", commandstr = ""
string checkcell
variable deletecol = 0
variable comp1 = 0, comp2 = 0
//search by choice in dropdown menu
for(n=0; n<SearchPanelNum; n+=1)
   inputvar = "SearchPanelInput" + num2istr(n)
   SVAR inputtype = $inputvar
   inputvar = "V_from" + num2istr(n)
   SVAR inputfrom = $inputvar
   inputvar = "V_to" + num2istr(n)
   SVAR inputto = $inputvar
   if(StringMatch(inputto, "the same"))
      inputto = inputfrom //set "To" default to be equal to "From" input
   endif
   if(StringMatch(inputtype, "Date")) //non-generic parsing to compare dates
      FindValue/TEXT=("Date")/TXOP=4 ExperimentalDetailsName
      if(V_value > -1)
         string inmonth, inday, inyear, checkmonth, checkday, checkyear
         variable indate, checkdate
         //check that input date is of correct format
         if(GrepString(inputfrom, "^\w{3}\s\d{2},\s\d{4}$"))
            sscanf inputfrom, "%3s %2s, %4s", inmonth, inday, inyear
            //convert date to unique number
            indate = 10000*str2num(inyear)+100*MonthToNum(inmonth)+str2num(inday)
            for(m=2; m<DimSize(ExperimentalDetailsSearch, 1); m+=1)
               checkcell = ExperimentalDetailsSearch[V_value][m]
               sscanf checkcell[5,16], "%3s %2s, %4s", checkmonth, checkday, checkyear
               checkdate = 10000*str2num(checkyear)+100*MonthToNum(checkmonth)+str2num(checkday)
               if(checkdate<indate)
                  DeletePoints/M=1 m, 1, ExperimentalDetailsSearch //delete nonmatching columns
                  m -= 1
               endif
            endfor
         else
            Print "\"Date From\" input not in correct format: example: Nov 02, 2011."
            Print "Not enforcing lower bound. (GetTable)"
         endif
         //could collapse 'to' loop with 'from' loop if desired
         if(GrepString(inputto, "^\w{3}\s\d{2},\s\d{4}$"))
            sscanf inputto, "%3s %2s, %4s", inmonth, inday, inyear
            indate = 10000*str2num(inyear)+100*MonthToNum(inmonth)+str2num(inday)
            for(m=2; m<DimSize(ExperimentalDetailsSearch, 1); m+=1)
               checkcell = ExperimentalDetailsSearch[V_value][m]
               sscanf checkcell[5,16], "%3s %2s, %4s", checkmonth, checkday, checkyear
               checkdate = 10000*str2num(checkyear)+100*MonthToNum(checkmonth)+str2num(checkday)
               if(checkdate>indate)
                  DeletePoints/M=1 m, 1, ExperimentalDetailsSearch
                  m -= 1
               endif
            endfor
         else
            Print "\"Date To\" input not in correct format: example: Nov 02, 2011."
            Print "Not enforcing upper bound. (GetTable)"
         endif
      else
         Print "\"Date\" not found in ExperimentalDetails.  Unable to search by date. (GetTable)"
      endif
   elseif(StringMatch(inputtype, "Time")) //compare times by hour
      FindValue/TEXT=("StartTime")/TXOP=4 ExperimentalDetailsName
```

```
   if(V_value > -1)
      variable checkhour
      for(m=2; m<DimSize(ExperimentalDetailsSearch, 1); m+=1)
         checkcell = ExperimentalDetailsSearch[V_value][m]
         checkhour = str2num(StringFromList(0, checkcell, ":"))
         if(StringMatch(checkcell[strlen(checkcell)-2,Inf], "PM"))
            checkhour += 12 //input time format is 24 hour
         endif
         if((checkhour<str2num(inputfrom)) || (checkhour > str2num(inputto)))
            DeletePoints/M=1 m, 1, ExperimentalDetailsSearch
            m -= 1
         endif
      endfor
      else
      Print "\StartTime\" not found in ExperimentalDetails."
      Print "Unable to search by time. (GetTable)"
   endif
elseif(StringMatch(inputtype, "VgStart")) //search all Vg#Start values for match
   make/O/N=1 GetTableVgIndices = -1
   k = 0
   do //find Vg#Start rows in ExperimentalDetails
      FindValue/TEXT=("Vg" + num2istr(k) + "Start")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         InsertPoints 0, 1, GetTableVgIndices
         GetTableVgIndices[0] = V_value //record row index
      endif
      k += 1
   while(V_value > -1)
   //search through columns as usual, comparing to each Vg#Start
   for(m=2; m<DimSize(ExperimentalDetailsSearch, 1); m+=1)
      deletecol = 1
      for(k=0; k<numpnts(GetTableVgIndices) - 1; k +=1)
         comp1 = str2num(ExperimentalDetailsSearch[GetTableVgIndices[k]][m])
         comp2 = str2num(ExperimentalDetailsSearch[k][m])
         if((comp1 >= str2num(inputfrom)) && (comp2 <= str2num(inputto)))
            deletecol = 0 //if any Vg#Start matches, don't delete the column
            break
         endif
      endfor
      if(deletecol == 1)
         DeletePoints/M=1 m, 1, ExperimentalDetailsSearch
         m -= 1
      endif
   endfor
elseif(StringMatch(inputtype, "VgStop")) //search all Vg#Stop values for match
   make/O/N=1 GetTableVgIndices = -1
   k = 0
   do
      FindValue/TEXT=("Vg" + num2istr(k) + "Stop")/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         InsertPoints 0, 1, GetTableVgIndices
         GetTableVgIndices[0] = V_value
      endif
      k += 1
   while(V_value > -1)
   for(m=2; m<DimSize(ExperimentalDetailsSearch, 1); m+=1)
      deletecol = 1
      for(k=0; k<numpnts(GetTableVgIndices) - 1; k +=1)
         comp1 = str2num(ExperimentalDetailsSearch[GetTableVgIndices[k]][m])
         comp2 = str2num(ExperimentalDetailsSearch[k][m])
         if((comp1 >= str2num(inputfrom)) && (comp2 <= str2num(inputto)))
            deletecol = 0
            break
```

```
            endif
         endfor
         if(deletecol == 1)
            DeletePoints/M=1 m, 1, ExperimentalDetailsSearch
            m -= 1
         endif
      endfor
   elseif(!StringMatch(inputtype, "-")) //default numerical comparison
      FindValue/TEXT=(inputtype)/TXOP=4 ExperimentalDetailsName
      if(V_Value > -1)
         for(m=2; m<DimSize(ExperimentalDetailsSearch, 1); m+=1)
            comp1 = str2num(ExperimentalDetailsSearch[V_value][m])
            comp2 = str2num(ExperimentalDetailsSearch[V_value][m])
            if((comp1 < str2num(inputfrom)) || (comp2 > str2num(inputto)))
               DeletePoints/M=1 m, 1, ExperimentalDetailsSearch
               m -= 1
            endif
         endfor
      else
         printf "\"%s\" not found in ExperimentalDetails.  ", inputtype
         printf "Unable to search. (GetTable)\r"
      endif
   endif
endfor

//search for specified dependent variables
//search is inclusive: any matching variable will allow the column to be included
FindValue/TEXT=("Dependent")/TXOP=4 ExperimentalDetailsName
string fullstr= ""
If(V_value > -1)
   string desearchlist = ""
   if(SearchPanelBmeas == 1)
      desearchlist += "Bmeas;"
   endif
   if(SearchPanelRd == 1)
      desearchlist += "Rd;"
   endif
   if(SearchPanelRxx == 1)
      desearchlist += "Rxx;"
   endif
   if(SearchPanelRxy == 1)
      desearchlist += "Rxy;"
   endif
   if(SearchPanelOpen == 1)
      desearchlist += V_open //user input text should be in form "DeVar1;Devar2;..."
   endif

   if(strlen(desearchlist) > 0)
      variable numdesearch = ItemsInList(desearchlist, ";")
      for(m=2; m<DimSize(ExperimentalDetailsSearch, 1); m+=1)
         deletecol = 1
         for(n=0; n<numdesearch; n+=1)
            fullstr = ExperimentalDetailsSearch[V_value][m]
            if(strsearch(fullstr, StringFromList(n, desearchlist) + ";", 0, 2) > -1)
               deletecol = 0 //if any input dependent variables match, column is not deleted
               break
            endif
         endfor
         if(deletecol == 1)
            DeletePoints/M=1 m, 1, ExperimentalDetailsSearch
            m -= 1
         endif
      endfor
```

```
   endif
else
   Print "\"Dependent\" not found in ExperimentalDetails."
   Print "Unable to search by dependent variable. (GetTable)"
endif

//search for specified independent variables
//search is exclusive: must match both variables (if specified)
FindValue/TEXT=("Independent")/TXOP=4 ExperimentalDetailsName
If(V_value > -1)
   string insearchlist1 = ""
   string insearchlist2 = ""
   ControlInfo invar1
   if(!(stringmatch(S_value, "-") || stringmatch(S_value, "InVar 1")))
      insearchlist1 += "(?i)" + S_value + "(\d{1,}[[:alpha:]]{0,1}){0,1}"
   endif
   ControlInfo invar2
   if(!(stringmatch(S_value, "-") || stringmatch(S_value, "InVar 2")))
      insearchlist2 += "(?i)" + S_value + "(\d{1,}[[:alpha:]]{0,1}){0,1}"
   endif

   string insearchlist = ""
   if(strlen(insearchlist1 + insearchlist2) > 0)
      FindValue/TEXT=("Independent")/TXOP=4 ExperimentalDetailsName
      NVAR SearchPanelInVarSym
      if(SearchPanelInVarSym == 0) //search for invar in order listed only
         if((strlen(insearchlist1) > 0) && (strlen(insearchlist2) > 0))
            insearchlist = "^" + insearchlist1 + ";" + insearchlist2 + "$"
         else
            insearchlist = "^" + insearchlist1 + insearchlist2 + "$"
         endif
         for(m=2; m<DimSize(ExperimentalDetailsSearch, 1); m+=1)
            deletecol = 1
            if(GrepString(ExperimentalDetailsSearch[V_value][m], insearchlist))
               deletecol = 0 //if all input independent variables match, column is not deleted
            endif
            if(deletecol == 1)
               DeletePoints/M=1 m, 1, ExperimentalDetailsSearch
               m -= 1
            endif
         endfor
      elseif(SearchPanelInVarSym == 1) //search for invar in any order
         for(m=2; m<DimSize(ExperimentalDetailsSearch, 1); m+=1)
            deletecol = 1
            fullstr = ExperimentalDetailsSearch[V_value][m]
            if(GrepString(fullstr, insearchlist1) && GrepString(fullstr, insearchlist2))
               deletecol = 0 //if all input independent variables match, column is not deleted
            endif
            if(deletecol == 1)
               DeletePoints/M=1 m, 1, ExperimentalDetailsSearch
               m -= 1
            endif
         endfor
      else
         printf "SearchPanelInVarSym value %g is not valid.  ", SearchPanelInVarSym
         printf "Must be 0 or 1.  Not searching by independent variable. (GetTable).\r"
      endif
   endif
else
   Print "\"Independent\" not found in ExperimentalDetails."
   Print "Unable to search by independent variable. (GetTable)"
endif
```

```
//add the 'name' column back onto the end of ExperimentalDetailsSave
Concatenate/T {ExperimentalDetailsName}, ExperimentalDetailsSearch

//check for rows desired for output
if(V_all != 1)
   string keeprows = "" //list of desired row names
   keeprows += "Date;"
   keeprows += "StartTime;"
   keeprows += "Index;"
   keeprows += "Dependent;"
   keeprows += "Independent;"
   keeprows += "DMMLIPA;"

   if(V_numpoints == 1)
      keeprows += "NumPoints X;"
      keeprows += "NumPoints Y;"
   endif

   if(V_ivsource == 1)
      keeprows += "33220A Freq;"
      keeprows += "SampleVac;"
      keeprows += "VacDivider;"
      keeprows += "VdcDivider;"
      keeprows += "SampleIac;"
      keeprows += "VtoI R;"
   endif

   if(V_scalefactors == 1)
      keeprows += "DMMoffset1;"
      keeprows += "DMMoffset2;"
      keeprows += "DMMoffset3;"
      keeprows += "DMMoffset4;"
      keeprows += "DMMscale1;"
      keeprows += "DMMscale2;"
      keeprows += "DMMscale3;"
      keeprows += "DMMscale4;"
   endif

   if(V_lipa == 1)
      keeprows += "LIsens1;"
      keeprows += "LItau1;"
      keeprows += "LIsens2;"
      keeprows += "LItau2;"
      keeprows += "PAamp1;"
      keeprows += "PAamp2;"
   endif
   if(V_temp == 1)
      keeprows += "Temp;"
      keeprows += "AV47Range;"
   endif

   //independent variables
   //if the 'all' box if checked, all related rows are displayed
   //if 'all' is not checked, but the "Info" box is: display Start, Stop, and Rate rows
   if(V_binfoall != 1)
      if(V_binfo == 1)
         keeprows += "BStart;"
         keeprows += "BStop;"
         keeprows += "BRate;"
      endif
   else
      keeprows += "BStart;"
      keeprows += "BStop;"
```

```
   keeprows += "BRate;"
   keeprows += "BMax;"
   keeprows += "BMin;"
   keeprows += "BRateDefault;"
   keeprows += "BRateMax;"
endif

if(V_idcinfoall != 1)
   if(V_idcinfo == 1)
      keeprows += "IdcStart;"
      keeprows += "IdcStop;"
      keeprows += "IdcRate;"
      keeprows += "VdcStart;"
      keeprows += "VdcStop;"
      keeprows += "VdcRate;"
   endif
else
   keeprows += "IdcStart;"
   keeprows += "IdcStop;"
   keeprows += "IdcRate;"
   keeprows += "IdcMax;"
   keeprows += "IdcMin;"
   keeprows += "IdcRateDefault;"
   keeprows += "IdcRateMax;"
   keeprows += "VdcStart;"
   keeprows += "VdcStop;"
   keeprows += "VdcRate;"
   keeprows += "VdcMax;"
   keeprows += "VdcMin;"
   keeprows += "VdcRateDefault;"
   keeprows += "VdcRateMax;"
endif

if(V_timeall != 1)
   if(V_timeinfo == 1)
      keeprows += "TimeStart;"
      keeprows += "TimeStop;"
      keeprows += "TimeRate;"
   endif
else
   keeprows += "TimeStart;"
   keeprows += "TimeStop;"
   keeprows += "TimeRate;"
   keeprows += "TimeMax;"
   keeprows += "TimeMin;"
   keeprows += "TimeRateDefault;"
   keeprows += "TimeRateMax;"
endif

//parse user input for Vg rows
for(n=0; n<strlen(V_vgnums); n+=1)
   //for each digit in user input keep the corresponding Vg values
   if(GrepString(V_vgnums[n], "\d"))
      if(V_vginfoall != 1)
         if(V_vginfo == 1)
            keeprows += "Vg" + V_vgnums[n] + "Start;"
            keeprows += "Vg" + V_vgnums[n] + "Stop;"
            keeprows += "Vg" + V_vgnums[n] + "Rate;"
         endif
      else
         keeprows += "Vg" + V_vgnums[n] + "Divider;"
         keeprows += "Vg" + V_vgnums[n] + "Start;"
         keeprows += "Vg" + V_vgnums[n] + "Stop;"
```

```
               keeprows += "Vg" + V_vgnums[n] + "Rate;"
               keeprows += "Vg" + V_vgnums[n] + "HardMax;"
               keeprows += "Vg" + V_vgnums[n] + "HardMin;"
               keeprows += "Vg" + V_vgnums[n] + "ExpMax;"
               keeprows += "Vg" + V_vgnums[n] + "ExpMin;"
               keeprows += "Vg" + V_vgnums[n] + "RateDefault;"
               keeprows += "Vg" + V_vgnums[n] + "RateMax;"
            endif
         endif
      endfor

      //remove unwanted rows
      for(m=0; m<DimSize(ExperimentalDetailsSearch, 0); m+=1)
         if(!StringMatch(keeprows, "*" + ExperimentalDetailsSearch[m][0] + ";*"))
            DeletePoints/M=0 m, 1, ExperimentalDetailsSearch
            m -= 1
         endif
      endfor
   endif

   //display output in table
   DoWindow SearchDisplay
   if(V_flag == 0)
      Edit/N=SearchDisplay ExperimentalDetailsSearch
   else
      DoWindow/F SearchDisplay
   endif
End

Function MonthToNum(name) //converts three letter month abbreviations to numbers
   string name

   strswitch(name)   // string switch
   case "Jan":
      return 1
   case "Feb":
      return 2
   case "Mar":
      return 3
   case "Apr":
      return 4
   case "May":
      return 5
   case "Jun":
      return 6
   case "Jul":
      return 7
   case "Aug":
      return 8
   case "Sep":
      return 9
   case "Oct":
      return 10
   case "Nov":
      return 11
   case "Dec":
      return 12
   endswitch
end
```

C.10 Physical Constants

These are functions which return various physical constants to make performing calculations easier. We include these primarily because some are used in various other procedures.

```
#pragma rtGlobals=1     // Use modern global access method.


//------------------------------------------------------------
//  A series of constants, Functions Start From C*, Unit in SI
//------------------------------------------------------------

Function Cc()
   //Speed of light in Vacuum
   return 299792458 // m s-1
end

Function Cmu0()
   //Permeability of free space
   return 12.566370614E-7 //N A-2 or H m-1
end

Function Cepsilon0()
   //Permittivity of free space
   return 8.854187817E-12 //A s V-1 m-1 or F m-1
end

Function Ch()
   //Planck's constant
   return 6.62606957E-34 //J s
end

Function Chbar()
   //Planck's constant
   return 1.054571726E-34 //J s
end

Function Ce()
   //elementary charge
   return  1.602176565E-19 //C
end

Function Calpha()
   //fine-structure constant
   return 7.2973525698E-3 //
end

Function CNA()
   //Avogadro's constant
   return 6.02214129E23 //
end

Function CR()
   //Gas constant
   return 8.3144621 // J mol-1 K-1
end
```

```
Function CKb()
   //Boltzmann's constant
   return 1.3806488E-23 //J K-1
end

Function CVm()
   //molar volume
   return 22.413968E-3 //m3 mol-1
end

Function Cme()
   //electron mass
   return 9.10938291E-31 //kg
end

Function CGaAsme()
   //effective electron mass in GaAs
   return 0.067 * Cme() //kg
end

Function Cmp()
   // proton mass
   return 1.672621777E-27 //kg
end

Function Cmn()
   // neutron mass
   return 1.674927351E-27 //kg
end

Function Ca0()
   // Bohr radius
   return 0.52917721092E-10 //m
end

Function Crc()
   // classical electron radius
   return 2.8179403267E-15 //m
end

Function Clambdac()
   // Compton wavelength
   return 2.4263102389E-12 //m
end

Function CmuB()
   // Bohr magneton
   return 927.400968E-26 //J T-1
end

Function CmuE()
   // electron magneton
   return -928.476430E-26 //J T-1
end

Function CmuNu()
   // nuclear magneton
   return 5.05078353E-27 //J T-1
end

Function CmuP()
   // proton magneton
   return 1.410606743E-26 //J T-1
```

```
end

Function CmuN()
   // neutron magneton
   return 0.96623647E-26 //J T-1
end

Function CsigmaT()
   // Thomson cross section
   return 0.6652458734E-28 //m2
end

Function CGN()
   // gravity constant
   return 6.67384E-11 //m3 kg-1 s-2
end

Function CTtr()
   // water triple point
   return 273.16 // K
end

Function CR0()
   // resistance quantum
   return 2*12906.4037217 //Omega
end

Function CG0()
   // conductance quantum
   return 1/Cr0() //S
   //7.7480917346E-5 / 2
end
```

C.11 Data Analysis

These are functions which have proven to be commonly useful in data analysis. `Split2D`, `Split2DCombo`, and `Waterfall` are helpful when working with two-dimensional waves. `HallGraph` plots all measured resistances of a single wave index on one graph as a function of magnetic field (refer to Section C.1 for a description of how magnetic field sweeps differ from other measurements).

```
#pragma rtGlobals=1     // Use modern global access method.

//create two 2D waves each with either only the odd columns
//or only the even columns of the original wave
function split2D(wname)
   wave wname

   variable numcol = dimsize(wname, 0)
   string wavestr = NameOfWave(wname)
```

```
   if(numcol < 2)
      printf "Wave %s is only 1D.  Not splitting. (split2D)\r", wavestr
      return -1
   endif

   //update InVarExp and DeVarExp to match current independent and dependent variable lists
   UpdateVariableLists()
   SVAR DeVarExp, InVarExp
   string matchexp = "(?i)((" + DeVarExp + ")(_(" + InVarExp + ")){2}_\d+)"
   if(GrepString(wavestr, matchexp))
      string devar = StringFromList(0, wavestr, "_")
      string indexnum = StringFromList(3, wavestr, "_")
      wavestr = devar + indexnum
   endif
   duplicate wname $(wavestr + "o") $(wavestr + "e")

   variable rowdelta = DimDelta(wname, 0)
   variable rowstart = DimOffset(wname, 0)
   variable n = numcol -1
   for(n=(numcol - 1); n>-1; n-=1)
      if(mod(n, 2) == 0)
         DeletePoints/M=0 n, 1, $(wavestr + "o")
      else
         DeletePoints/M=0 n, 1, $(wavestr + "e")
      endif
   endfor
   SetScale/P x, rowstart, rowdelta*2, $(wavestr + "e")
   SetScale/P x, rowstart + rowdelta, rowdelta*2, $(wavestr + "o")
end

//create 1D cuts of a 2D wave with a specified offset between consecutive cuts
function waterfall(wname, direction, offset, [shift, makegraph])
   wave wname
   string direction //"v": vertical, "h": horizontal
   variable offset //offset between consecutive cuts
   variable shift //overall shift added to all cuts
   variable makegraph //"1": display resulting cuts in new graph

   if(ParamIsDefault(shift))
      shift = 0
   endif
   if(ParamIsDefault(makegraph))
      makegraph = 0
   endif

   variable numcol = dimsize(wname, 1)
   variable numrow = dimsize(wname, 0)
   string wavestr = NameOfWave(wname)
   if(numcol < 2)
      printf "Wave %s is only 1D.  Not making waterfall. (waterfall)\r", wavestr
      return -1
   endif
   if(dimsize(wname, 2) > 1)
      printf "Wave %s has 3 or more dimensions.  Not making waterfall. (waterfall)\r", wavestr
      return -1
   endif

   //update InVarExp and DeVarExp to match current independent and dependent variable lists
   UpdateVariableLists()
   SVAR DeVarExp, InVarExp
   string matchexp = "(?i)((" + DeVarExp + ")(_(" + InVarExp + ")){2}_\d+)"
   if(GrepString(wavestr, matchexp))
      string devar = StringFromList(0, wavestr, "_")
```

```
      string indexnum = StringFromList(3, wavestr, "_")
      wavestr = devar + indexnum
   endif

   variable n = 0
   if(StringMatch(direction, "h"))
      if(makegraph == 1)
         display
      endif
      for(n=0; n<numcol; n+=1)
         duplicate/O/R=[][n] wname $(wavestr + "_h" + num2istr(n))
         wave waterfallworking = $(wavestr + "_h" + num2istr(n))
         waterfallworking += offset * n + shift
         if(makegraph == 1)
            AppendToGraph $(wavestr + "_h" + num2istr(n))
         endif
      endfor
   elseif(StringMatch(direction, "v"))
      variable coldelta = DimDelta(wname, 1)
      variable colstart = DimOffset(wname, 1)
      if(makegraph == 1)
         display
      endif
      for(n=0; n<numrow; n+=1)
         duplicate/O/R=[n][] wname $(wavestr + "_v" + num2istr(n))
         wave waterfallworking = $(wavestr + "_v" + num2istr(n))
         redimension/N=(numcol) waterfallworking
         SetScale/P x, colstart, coldelta, waterfallworking
         waterfallworking += offset * n + shift
         if(makegraph ==1)
            AppendToGraph $(wavestr + "_v" + num2istr(n))
         endif
      endfor
   else
      printf "Direction \"%s\" not valid.  Please use \"h\" or \"v\". (waterfall)\r", direction
   endif
end

function split2dcombo(wname) //split an Idc 2d plot with even and odd sweep direction
   string wname

   //split the 2d data to two new 2d waves
   split2d($wname)
   string wavestr=wname
   //update InVarExp and DeVarExp to match current independent and dependent variable lists
   UpdateVariableLists()
   SVAR DeVarExp, InVarExp
   string matchexp = "(?i)((" + DeVarExp + ")(_(" + InVarExp + ")){2}_\d+)"
   if(GrepString(wavestr, matchexp))
      string devar = StringFromList(0, wavestr, "_")
      string indexnum = StringFromList(3, wavestr, "_")
      wavestr = devar + indexnum
   endif

   //plot two new 2d plot with only odd and only even directions.
   display /N=$(wavestr+"o")
   appendimage $(wavestr+"o")
   movewindow/ w=$(wavestr+"o0") 0, 0, 400,200
   display /N=$(wavestr+"e")
   appendimage $(wavestr+"e")
   movewindow/ w=$(wavestr+"e0") 400, 0, 800,200
   ModifyImage/w=$(wavestr+"o0") $(wavestr+"o") ctab= {*,*,Terrain,0}
   ModifyImage/w=$(wavestr+"e0") $(wavestr+"e") ctab= {*,*,Terrain,0}
```

```
   //plot two 1d graphs with odd traces and even traces
   waterfall($(wavestr+"e"), "v", 0, makegraph = 1)
   movewindow/ w=$(WinName(0,1)) 0, 300, 400, 500
   waterfall($(wavestr+"o"), "v", 0, makegraph = 1)
   movewindow/ w=$(WinName(0,1)) 400, 300, 800, 500
end

//plot all resistance waves of the specified index against B in one graph
function HallGraph(wavenum)
   variable wavenum

   string RxxWave = "Rxx_time_" + num2istr(wavenum)
   string RxyWave = "Rxy_time_" + num2istr(wavenum)
   string RdWave = "Rd_time_" + num2istr(wavenum)
   string RdpWave = "Rdp_time_" + num2istr(wavenum)
   string RdnWave = "Rdm_time_" + num2istr(wavenum)
   string BWave = "Bmeas_time_" + num2istr(wavenum)

   if(!WaveExists($Bwave))
      printf "Wave %s does not exist; cannot graph.  Aborting. (HallGraph)\r", BWave
      Abort "HallGraph: no Bmeas wave."
   endif

   string rightlabel = ""
   display
   if(WaveExists($RxxWave))
      AppendtoGraph $RxxWave vs $BWave
      ModifyGraph axRGB(left)=(65280,0,0)
      Label left "Rxx (h/e\\S2\\M)"
      ModifyGraph axOffset(left)=-1.7
   endif
   if(WaveExists($RxyWave))
      AppendtoGraph/r $RxyWave vs $BWave
      ModifyGraph rgb($RxyWave)=(0,0,0)
      rightlabel += "Rxy, "
   endif
   if(WaveExists($RdWave))
      AppendtoGraph/r $RdWave vs $BWave
      ModifyGraph rgb($RdWave)=(0,0,52224)
      rightlabel += "Rd, "
   endif
   if(WaveExists($RdpWave))
      AppendtoGraph/r $RdpWave vs $BWave
      ModifyGraph rgb($RdpWave)=(0,0,52224)
      rightlabel += "Rdp, "
   endif
   if(WaveExists($RdnWave))
      AppendtoGraph/r $RdnWave vs $BWave
      ModifyGraph rgb($RdnWave)=(0,0,52224)
      rightlabel += "Rdn, "
   endif

   Label bottom "B (T)"
   variable labellength = strlen(rightlabel)
   if(labellength>0)
      ModifyGraph axOffset(right)=-1.2
      rightlabel = rightlabel[0, (labellength - 3)] + " (h/e\\S2\\M)"
      Label right rightlabel
   endif
   Legend/C/N=text0/F=0/B=1/A=LT/X=0.00/Y=0.00
end
```

C.12 Fit Functions

These are functions used in IGOR's least-squares fitting routine. `CoulombBlockade` is used to fit to a thermally-broadened Coublomb blockade peak to extract electron temperature. `TunnelingConductance` and the related functions are used to fit the differential tunneling conductance to r_D, discussed in Section 3.2. The remaining functions relate to fits performed to estimate the QPC width, as described in Section 2.3.

```
#pragma rtGlobals=1     // Use modern global access method.

function CoublombBlockade(w, x) : FitFunc
   Wave w
   variable x //gate voltage in mV

   //CurveFitDialog/ Coefficients 5
   //CurveFitDialog/ w[0] = T (mK)
   //CurveFitDialog/ w[1] = G_0 //amplitude coefficient
   //CurveFitDialog/ w[2] = x_0
   //CurveFitDialog/ w[3] = y_0
   //CurveFitDialog/ w[4] = alpha_G //gate lever arm

   return Ch()*w[1]/(4*CkB()*w[0]*1e-3*(cosh(-Ce()*w[4]*(x-w[2])*1e-3/(2*CkB()*w[0]*1e-3)))^2) + w[3]
end

Function TunnelingConductance(w,x) : FitFunc
   Wave w
   Variable x

   //CurveFitDialog/ These comments were created by the Curve Fitting dialog. Altering them will
   //CurveFitDialog/ make the function less convenient to work with in the Curve Fitting dialog.
   //CurveFitDialog/ Equation:
   //CurveFitDialog/ f(x) = y_0+ A*BasFgFunc(w*(x-x_0),g)
   //CurveFitDialog/ End of Equation
   //CurveFitDialog/ Independent Variables 1
   //CurveFitDialog/ x
   //CurveFitDialog/ Coefficients 6
   //CurveFitDialog/ w[0] = y_0
   //CurveFitDialog/ w[1] = x_0
   //CurveFitDialog/ w[2] = A
   //CurveFitDialog/ w[3] = g
   //CurveFitDialog/ w[4] = estar
   //CurveFitDialog/ w[5] = T (mK)

   variable kT=CkB()*w[5]/1000
   variable factor=w[4]*Ce()*1e-9*CR0()/((5/2)*kT)

   return w[0]+ w[2]*(w[5]/1000)^(2*w[3]-2)*BasFgFunc(factor*(x-w[1]),w[3])
End

Function BasFgFunc(y,g)
   Variable y,g
```

```
   Variable/C cp1,cp2,cBetaRes, cnormal, cp3
   cp1= cmplx(g,y/(2*PI))
   cp2= cmplx(g,-1*y/(2*PI))
   cp3=cmplx(g,0)

   cnormal=EulerB(cp3,cp3)
   cBetaRes= EulerB(cp1,cp2)
   if ( abs(imag(cBetaRes)) > 1e-8 )
      print "Warning, Bas's function is giving an imaginary EulerB"
      print cBetaRes
   endif

   Variable p3, normal
   p3= 2*sinh(y/2)*imag(digamma(cp1))
   normal=Real(cnormal)/pi

   Variable result
   result= real(cBetaRes)*(PI*cosh(y/2)-p3)/normal
   return result
End

Function/C EulerB(z1,z2)
   Variable/C z1,z2

   Variable/C beta
   beta= gamma(z1)*gamma(z2)/gamma(z1+z2)
   return beta
End

function RdQPCSquareLowB(w, x) : FitFunc
   wave w
   variable x

   //CurveFitDialog/ Independent Variables 1
   //CurveFitDialog/ B
   //CurveFitDialog/ Coefficients 3
   //CurveFitDialog/ w[0] = Wnm
   //CurveFitDialog/ w[1] = npercm10
   //CurveFitDialog/ w[2] = offset

   variable width = w[0] * 1e-9
   variable nden = w[1] * 1e14
   variable arg = width*Ce()*x/(2*Chbar()*sqrt(2*pi*nden))
   variable Nqpc = ((2*nden*Chbar())/(Ce()*x))*(asin(arg)+arg*sqrt(1-arg^2))
   return 1/Nqpc/2 + w[2]
end

function RdQPCSquareHighB(w, x) : FitFunc
   wave w
   variable x

   //CurveFitDialog/ Independent Variables 1
   //CurveFitDialog/ B
   //CurveFitDialog/ Coefficients 3
   //CurveFitDialog/ w[0] = Wnm
   //CurveFitDialog/ w[1] = npercm10
   //CurveFitDialog/ w[2] = offset

   variable width = w[0] * 1e-9
   variable nden = w[1] * 1e14
   variable Nqpc = (nden*Ch())/(2*Ce()*x)
   //variable Nqpc = nden*Ch()/(2*Ce()*x) + 1/2
   return 1/Nqpc/2 + w[2]
```

```
end

function RdQPCSquareWell(w, x) : FitFunc
   wave w
   variable x

   //CurveFitDialog/ Independent Variables 1
   //CurveFitDialog/ B
   //CurveFitDialog/ Coefficients 3
   //CurveFitDialog/ w[0] = Wnm
   //CurveFitDialog/ w[1] = npercm10
   //CurveFitDialog/ w[2] = offset

   variable width = w[0] * 1e-9
   variable nden = w[1] * 1e14
   variable arg = width*Ce()*x/(2*Chbar()*sqrt(2*pi*nden))
   variable Nqpc
   if(arg < 1)
      Nqpc = ((2*nden*Chbar())/(Ce()*x))*(asin(arg)+arg*sqrt(1-arg^2))
   else
      Nqpc = (nden*Ch())/(2*Ce()*x)
      //Nqpc = nden*Ch()/(2*Ce()*x) + 1/2
   endif
   return 1/Nqpc/2 + w[2]
end

function RdQPCParabolic(w, x) : FitFunc
   wave w
   variable x

   //CurveFitDialog/ Independent Variables 1
   //CurveFitDialog/ B
   //CurveFitDialog/ Coefficients 3
   //CurveFitDialog/ w[0] = Wnm
   //CurveFitDialog/ w[1] = npercm10
   //CurveFitDialog/ w[3] = offset

   variable width = w[0] * 1e-9
   variable nden = w[1] * 1e14
   variable Nqpc = nden*pi*Chbar()/sqrt((Ce()*x)^2 + 2*pi*nden*(2*Chbar()/width)^2)
   //variable Nqpc = nden*pi*Chbar()/sqrt((Ce()*x)^2 + 2*pi*nden*(2*Chbar()/width)^2) + 1/2
   return 1/Nqpc/2 + w[2]
end
```

Bibliography

[1] X.-G. Wen. Phys. Rev. B **44**, 5708 (1991).

[2] X. G. Wen. Phys. Rev. B **41**, 12838 (1990).

[3] B. I. Halperin. Phys. Rev. Lett. **52**, 1583 (1984).

[4] C. Nayak, S. H. Simon, A. Stern, M. Freedman, and S. Das Sarma. Rev. Mod. Phys. **80**, 1083 (2008).

[5] M. Dolev, M. Heiblum, V. Umansky, A. Stern, and D. Mahalu. Nature (London) **452**, 829 (2008).

[6] V. J. Goldman and B. Su. Science **267**, 1010 (1995).

[7] F. E. Camino, W. Zhou, and V. J. Goldman. Phys. Rev. Lett. **98**, 076805 (2007).

[8] B. I. Halperin, A. Stern, I. Neder, and B. Rosenow. Phys. Rev. B **83**, 155440 (2011).

[9] A. Stern and B. I. Halperin. Phys. Rev. Lett. **96**, 016802 (2006).

[10] R. L. Willett, L. N. Pfeiffer, and K. W. West. Proc. Nat. Acad. Sci. USA **106**, 8853 (2009).

[11] R. L. Willett, L. N. Pfeiffer, and K. W. West. Phys. Rev. B **82**, 205301 (2010).

[12] A. Bid, N. Ofek, H. Inoue, M. Heiblum, C. L. Kane, V. Umansky, and D. Mahalu. Nature (London) **466**, 585 (2010).

[13] V. Venkatachalam, S. Hart, L. Pfeiffer, K. West, and A. Yacoby. e-print `arXiv:1202.6681v1` (2012).

[14] G. Nachtwei. Physica E (Amsterdam) **4**, 79 (1999).

[15] K. von Klitzing, G. Dorda, and M. Pepper. Phys. Rev. Lett. **45**, 494 (1980).

[16] H. L. Stormer. Rev. Mod. Phys. **71**, 875 (1999).

[17] A. Usher, R. J. Nicholas, J. J. Harris, and C. T. Foxon. Phys. Rev. B **41**, 1129 (1990).

[18] D. C. Tsui, H. L. Stormer, and A. C. Gossard. Phys. Rev. Lett. **48**, 1559 (1982).

[19] R. B. Laughlin. Phys. Rev. Lett. **50**, 1395 (1983).

[20] R. de Picciotto, M. Reznikov, M. Heiblum, V. Umansky, G. Bunin, and D. Mahalu. Nature (London) **389**, 162 (1997).

[21] L. Saminadayar, D. C. Glattli, Y. Jin, and B. Etienne. Phys. Rev. Lett. **79**, 2526 (1997).

[22] F. D. M. Haldane. Phys. Rev. Lett. **51**, 605 (1983).

[23] B. I. Halperin. Phys. Rev. Lett. **52**, 1583 (1984).

[24] R. B. Laughlin. Surf. Sci. **142**, 163 (1984).

[25] J. K. Jain. Phys. Rev. Lett. **63**, 199 (1989).

[26] J. K. Jain and V. J. Goldman. Phys. Rev. B **45**, 1255 (1992).

[27] H. L. Stormer, D. C. Tsui, and A. C. Gossard. Rev. Mod. Phys. **71**, S298 (1999).

[28] R. Willett, J. P. Eisenstein, H. L. Störmer, D. C. Tsui, A. C. Gossard, and J. H. English. Phys. Rev. Lett. **59**, 1776 (1987).

[29] C. R. Dean, B. A. Piot, P. Hayden, S. Das Sarma, G. Gervais, L. N. Pfeiffer, and K. W. West. Phys. Rev. Lett. **100**, 146803 (2008).

[30] Y. W. Suen, L. W. Engel, M. B. Santos, M. Shayegan, and D. C. Tsui. Phys. Rev. Lett. **68**, 1379 (1992).

[31] J. P. Eisenstein, G. S. Boebinger, L. N. Pfeiffer, K. W. West, and S. He. Phys. Rev. Lett. **68**, 1383 (1992).

[32] D. Arovas, J. R. Schrieffer, and Frank Wilczek. Phys. Rev. Lett. **53**, 722 (1984).

[33] C. Nayak and F. Wilczek. Nucl. Phys. B **479**, 529 (1996).

[34] A. Y. Kitaev. Ann. Phys. (NY) **303**, 2 (2003).

[35] F. D. M. Haldane. Phys. Rev. Lett. **51**, 605 (1983).

[36] F. D. M. Haldane and E. H. Rezayi. Phys. Rev. Lett. **60**, 956 (1998).

[37] A. Zaslavsky, D. A. Grutzmacher, S. Y. Lin, T. P. Smith III, R. A. Kiehl, and T. O. Sedgwick. Phys. Rev. B **47**, 16036 (1993).

[38] V. Melik-Alaverdian and N. E. Bonesteel. Phys. Rev. B **52**, R17032 (1995).

[39] A. Wójs, C. Tőke, and J. K. Jain. Phys. Rev. Lett. **105**, 096802 (2010).

[40] B. I. Halperin. Phys. Rev. B **25**, 2185 (1982).

[41] C. de C. Chamon and X. G. Wen. Phys. Rev. B **49**, 8227 (1994).

[42] A. H. MacDonald. Phys. Rev. Lett. **64**, 220 (1990).

[43] X. G. Wen. Phys. Rev. Lett. **64**, 2206 (1990).

[44] C. L. Kane, M. P. A. Fisher, and J. Polchinski. Phys. Rev. Lett. **72**, 4129 (1994).

[45] M. Büttiker. Phys. Rev. Lett. **57**, 1761 (1986).

[46] C. W. J. Beenakker and H. van Houten. Solid State Phys. **44**, 1 (1991).

[47] A. M. Chang. Rev. Mod. Phys. **75**, 1449 (2003).

[48] S. Adachi, A. R. Adams, M. Missous, F. H Pollak, and V. A. Wilkinson. *Properities of Aluminum Gallium Arsenide*. S. Adachi, editor. INSPEC, London (1993).

[49] E. F. Schubert, J. B. Stark, B. Ullrich, and J. E. Cunningham. Appl. Phys. Lett. **52**, 1508 (1988).

[50] I. P. Radu, J. B. Miller, C. M. Marcus, M. A. Kastner, L. N. Pfeiffer, and K. W. West. Science **320**, 899 (2008).

[51] U. Meirav, M. A. Kastner, and S. J. Wind. Phys. Rev. Lett. **65**, 771 (1990).

[52] J. R. Petta, A. C. Johnson, C. M. Marcus, M. P. Hanson, and A. C. Gossard. Phys. Rev. Lett. **93**, 186802 (2004).

[53] I. Radu. PhD thesis, Massachusetts Institute of Technology (2009).

[54] P. M. Mooney. J. Appl. Phys. **67**, R1 (1990).

[55] L. N. Pfieffer. Private Communication.

[56] R. J. Almassy, D. C. Reynolds, C. W Litton, K. K. Bajaj, and G. L. McCoy. Solid State Commun. **38**, 1053 (1981).

[57] G. Moore and N. Read. Nucl. Phys. B **360**, 362 (1991).

[58] X. G. Wen. Phys. Rev. Lett. **66**, 802 (1991).

[59] M. Levin, B. I. Halperin, and B. Rosenow. Phys. Rev. Lett. **99**, 236806 (2007).

[60] S.-S. Lee, S. Ryu, C. Nayak, and M. P. A. Fisher. Phys. Rev. Lett. **99**, 236807 (2007).

[61] P. Fendley, A. W. W. Ludwig, and H. Saleur. Phys. Rev. B **52**, 8934 (1995).

[62] S. Roddaro, V. Pellegrini, F. Beltram, G. Biasiol, and L. Sorba. Phys. Rev. Lett. **93**, 046801 (2004).

[63] S. Roddaro, V. Pellegrini, F. Beltram, L. N. Pfeiffer, and K. W. West. Phys. Rev. Lett. **95**, 156804 (2005).

[64] M. R. Peterson, K. Park, and S. Das Sarma. Phys. Rev. Lett. **101**, 156803 (2008).

[65] M. R. Peterson, T. Jolicoeur, and S. Das Sarma. Phys. Rev. Lett. **101**, 016807 (2008).

[66] X. Wan, Z.-X. Hu, E. H. Rezayi, and K. Yang. Phys. Rev. B **77**, 165316 (2008).

[67] A. E. Feiguin, E. Rezayi, K. Yang, C. Nayak, and S. Das Sarma. Phys. Rev. B **79**, 115322 (2009).

[68] H. Wang, D. N. Sheng, and F. D. M. Haldane. Phys. Rev. B **80**, 241311(R) (2009).

[69] S. Das Sarma, G. Gervais, and X. Zhou. Phys. Rev. B **82**, 115330 (2010).

[70] M. Storni, R. H. Morf, and S. Das Sarma. Phys. Rev. Lett. **104**, 076803 (2010).

[71] C. Wang and D. E. Feldman. Phys. Rev. B **82**, 165314 (2010).

[72] E. H. Rezayi and S. H. Simon. Phys. Rev. Lett. **106**, 116801 (2011).

[73] C. Zhang, T. Knuuttila, Y. Dai, R. R. Du, L. N. Pfeiffer, and K. W. West. Phys. Rev. Lett. **104**, 166801 (2010).

[74] M. Stern, P. Plochocka, V. Umansky, D. K. Maude, M. Potemski, and I. Bar-Joseph. Phys. Rev. Lett. **105**, 096801 (2010).

[75] V. Venkatachalam, A. Yacoby, L. N. Pfeiffer, and K. W. West. Nature (London) **469**, 185 (2011).

[76] G. Möller, A. Wójs, and N. R. Cooper. Phys. Rev. Lett. **107**, 036803 (2011).

[77] M. Stern, B. A. Piot, Y. Vardi, V. Umansky, P. Plochocka, D. K. Maude, and I. Bar-Joseph. Phys. Rev. Lett. **108**, 066810 (2012).

[78] R. H. Morf. Phys. Rev. Lett. **80**, 1505 (1998).

[79] B. I. Halperin. Helv. Phys. Acta. **56**, 75 (1983).

[80] B. I. Halperin, P. A. Lee, and N. Read. Phys. Rev. B **47**, 7312 (1993).

[81] X. G. Wen and Q. Niu. Phys. Rev. B **41**, 9377 (1990).

[82] W. Bishara, P. Bonderson, C. Nayak, K. Shtengel, and J. K. Slingerland. Phys. Rev. B **80**, 155303 (2009).

[83] P. Fendley. Private Communication.

[84] K. Yang. Phys. Rev. Lett. **91**, 036802 (2003).

[85] B. J. Overbosch and X.-G. Wen. e-print `arXiv:0804.2087v1` (2008).

[86] F. P. Milliken, C. P. Umbach, and R. A. Webb. Solid State Commun. **97**, 309 (1996).

[87] M. E. Cage, A. M. Chang, and S. M. Girvin. *The Quantum Hall Effect*. R. E. Prange and S. M Girvin, editors. Springer-Verlag, New York (1987).

[88] K. Yoshihiro, J. Kinoshita, K. Inagaki, C. Yamanouchi, J. Moriyama, and S. Kawaji. Surf. Sci. **113**, 16 (1982).

[89] G. Ebert, K. von Klitzing, K. Ploog, and G. Weimann. J. Phys. C **16**, 5441 (1983).

[90] T. Takamasu, H. Dodo, and N. Miura. Solid State Commun. **96**, 121 (1995).

[91] J. P. Watts, A. Usher, A. J. Matthews, M. Zhu, M. Elliott, W. G. Herrenden-Harker, P. R. Morris, M. Y. Simmons, and D. A. Ritchie. Phys. Rev. Lett. **81**, 4220 (1998).

[92] S. Kawaji, K. Hirakawa, M. Nagata, T. Okamoto, T. Fukase, and T. Gotoh. J. Phys. Soc. Jpn. **63**, 2303 (1994).

[93] A. Boisen, P. Bøggild, A. Kristensen, and P. E. Lindelof. Phys. Rev. B **50**, 1957 (1994).

[94] T. Okuno, S. Kawaji, T. Ohrui, T. Okamoto, Y. Kurata, and J. Sakai. J. Phys. Soc. Jpn. **64**, 1881 (1995).

[95] Y. Kawaguchi, F. Hayashi, S. Komiyama, T. Osada, Y. Shiraki, and R. Itoh. Jpn. J. of Appl. Phys. **34**, 4309 (1995).

[96] N. Q. Balaban, U. Meirav, and H. Shtrickman. Phys. Rev. B **52**, R5503 (1995).

[97] S. Kawaji. Semicond. Sci. Technol. **11**, 1546 (1996).

[98] T. Sanuki, K. Oto, S. Takaoka, K. Murase, and K. Gamo. Solid State Commun. **117**, 343 (2001).

[99] K. Oto, T. Sanuki, S. Takaoka, K. Murase, and K. Gamo. Physica E (Amsterdam) **12**, 173 (2002).

[100] N. Q. Balaban, U. Meirav, H. Shtrikman, and Y. Levinson. Phys. Rev. Lett. **71**, 1443 (1993).

[101] A. L. Efros. Solid State Commun. **67**, 1019 (1988).

[102] A. H. MacDonald, T. M. Rice, and W. F. Brinkman. Phys. Rev. B **28**, 3648 (1983).

[103] D. J. Thouless. J. Phys. C **18**, 6211 (1985).

[104] L. Eaves and F. W. Sheard. Semicond. Sci. Technol. **1**, 346 (1986).

[105] K. Shizuya. Phys. Rev. B **60**, 8218 (1999).

[106] G. Ebert, K. von Klitzing, K. Ploog, and G Weimann. J. Phys. C **16**, 5441 (1983).

[107] S. Komiyama, T. Takamasu, S. Hiyamizu, and S. Sasa. Solid State Commun. **54**, 479 (1985).

[108] H. Akera. J. Phys. Soc. Jpn. **69**, 3174 (2000).

[109] P. Středa and K. von Klitzing. J. Phys. C **17**, L483 (1984).

[110] V. Tsemekhman, K. Tsemekhman, C. Wexler, J. H. Han, and D. J. Thouless. Phys. Rev. B **55**, R10201 (1997).

[111] B. N. Taylor and T. J. Witt. Metrologia **26**, 47 (1989).

[112] N. G. Kalugin, B. E. Sağol, A. Buß, A. Hirsch, C. Stellmach, G. Hein, and G. Nachtwei. Phys. Rev. B **68**, 125313 (2003).

[113] L. Bliek, E. Braun, G. Hein, V. Kose, J. Niemeyer, G. Weimann, and W. Schlapp. Semicond. Sci. Technol. **1**, 110 (1986).

[114] J. R. Kirtley, Z. Schlesinger, T. N. Theis, F. P. Milliken, S. L. Wright, and L. F. Palmateer. Phys. Rev. B **34**, 1384 (1986).

[115] V. Singh and M. Deshmukh. Phys. Rev. B **80**, 081404(R) (2009).

[116] P. G. N. de Vegvar, A. M. Chang, G. Timp, P. M. Mankiewich, J. E. Cunningham, R. Behringer, and R. E. Howard. Phys. Rev. B **36**, 9366 (1987).

[117] G. Nachtwei, Z. H. Liu, G. Lütjering, R. R. Gerhardts, D. Weiss, K. von Klitzing, and K. Eberl. Phys. Rev. B **57**, 9937 (1998).

[118] H. Iizuka, S. Kawaji, and T. Okamoto. Physica E (Amsterdam) **6**, 132 (2000).

[119] G. Vasile, C. Stellmach, G. Hein, and G. Nachtwei. Semicond. Sci. Technol. **21**, 1714 (2006).

[120] J. K. Jain and R. K. Kamilla. *Composite Fermions*. O. Heinonen, editor. World Scientific, Singapore (1998).

[121] J. B. Miller. PhD thesis, Harvard University (2007).

[122] D. T. McClure. PhD thesis, Harvard University (2012).

[123] A. G. Baca and C. I. H. Ashby. *Fabrication of GaAs Devices*. Institution of Engineering and Technology, London (2005).